Universitatea *Transilvania* din Braşov
Facultatea de Matematică–Informatică

ERNEST SCHEIBER

PROGRAMARE DISTRIBUITĂ ÎN JAVA

Braşov

Cuprins

Introducere

Cursul *Programare distribuită în Java 2* este continuarea cursului *Programare distribuită în Java 1.* Minimal, pentru înţelegerea cursului este nevoie de cunoaşterea capitolelor *Servlet* şi *JSP* din partea întâi.

Tematica cursului cuprinde:

- Cadre de dezvoltare (framework) a aplicaţiilor Web:
  - *Struts2*
  - *Java Server Faces*
  - *Google Web Toolkit*
- Servicii Web cu:
  - Servicii RPC
    - *JAX-WS (Java API for XML Web Services)* prin *metro*
  - Servicii REST
    - *JAX-RS (Java API for RESTful Web Services)* prin *jersey*
- Modelul OSGi (Open Source Gateway initiative)

Produsele informatice utilizate

Pe durata existenţei, produsele informatice evoluează prin versiunile pe care producătorii ni le pun la dispoziţie. Nu de puţine ori o versiune nouă nu este compatibilă cu versiunea anterioară, fapt care necesită adaptarea programelor *client*.

Lista următoare precizează versiunile produselor utilizate în lucrare, indicate în majoritatea cazurilor prin resursa de instalare. Lista conţine doar produsele care completează pe cele utilizate în *Programare distribuită 1*.

Versiunile produselor informatice utilizate în lucrare		
No.	Produsul informatic	Resursa/versiunea
1	apache-karaf	apache-karaf-3.0.0.zip
2	equinox (OSGi)	org.eclipse.osgi_3.9.0.v20130529-1710.jar
3	Equinox Bridge Servlet	bridge.war
4	felix	org.apache.felix.main.distribution-4.2.1.tar.gz
5	Google Web Toolkit	gwt-2.5.1.zip
6	Google-gson	google-gson-2.2.4-release.zip
7	jersey	jaxrs-ri-2.5.zip
8	Java Server Faces (JSF)	javax.faces-2.2.4.jar
9	knopflerfish	knopflerfish_osgi_sdk_5.0.0.jar
10	metro	metro-standalone-2.3.zip
11	struts	struts2-2.3.16-all.zip

Partea I

APLICAŢII WEB CADRE DE LUCRU

Capitolul 1

Aplicaţii Web

Printre aplicaţiile distribuite de tip client server, în care comunicaţiile se bazează pe protocolul *http,* se disting

- *Aplicaţii Web (site):* Cererea adresată serverului este lansată de o persoană prin intermediul unui site, utilizând un program navigator: *Mozilla Firefox, Google Chrome, Microsoft InternetExplorer, Opera, Apple Safari, etc.*

- *Servicii Web:* Cererea către server se face de un program. Aplicaţia server şi client se programează utilizând interfeţe de programare specifice.

Sunt cunoscute multe cadre de dezvoltare (framework) a aplicaţiilor Web gratuite (http://java-source.net/open-source/web-frameworks):

1.1 Modelul MVC

Modelul **Model–View–Controller** (MVC) este o schemă de proiectare a aplicaţiilor client-server. Modelul MVC este alcătuită din trei părţi:

- Componenta *Model* conţine clasele ce rezolvă cererile clienţilor asigurând funcţionalitatea aplicaţiei (business classes).

- Componenta *View* asigură interfaţa grafică a clientului.

- Componenta *Controller* realizează comunicarea între clasele şi / sau resursele din *View* cu clasele din *Model.*

Bazate pe folosirea tehnologiei servlet-urilor şi a JSP, materializarea modelului MVC este bine reprezentată de cadrele de dezvoltare (framework) pentru aplicaţii Web:

- *Struts*
- *Java Server Faces (JSF)*
- *Google Web Toolkit* (GWT)

care sunt prezentate în continuare.

În fiecare caz, controlul activităţilor se face de către cadrul de lucru, deci are loc inversarea controlului.

1.2 Struts 2

Struts2 este un cadru de dezvoltare (framework) a aplicaţiilor bazat pe modelul MVC dezvoltat în cadrul proiectului *apache.*

Struts2 utilizează atât modelul descriptiv de aplicaţie, prin intermediul unui fişier `xml,` de configurare, `struts.xml,` cât şi modul programat prin adnotări.

Instalarea produsului. Fişierul *struts2-blank-2.*.*.war* obţinut din dezarhivarea fişierului descărcat de la *www.apache.org* oferă un cadru minimal pentru dezvoltarea unei aplicaţii. Acest fişier se copiază în catalogul `TOMCAT_HOME\webapps` şi în urma (re)pornirii serverului `tomcat,` fişierul este dezarhivat.

Alternativ, este suficient ca fişierele `jar` din catalogul

`struts2-blank-2.*.*\WEB-INF\lib`

să fie copiate în catalogul `lib` al aplicaţiei Web.

1.2.1 Aplicaţii *Struts2* prin modelul descriptiv

Modelul descriptiv este caracterizat print-un fişier de configurare `struts.xml.` Componentele unei aplicaţii *Struts2* sunt:

- Componenta *View* este alcătuită din documente `html` sau pagini `JSP,` conţinând formulare prin care clientul introduce date şi prin care se afişează rezultatele.

  Apelarea din exteriorul aplicaţiei *Struts2* se face prin

  `http://`*host:port/context/NumeAcţiune*`.action`

  iar din interiorul aplicaţiei prin *NumeAcţiune*`.action.`

- Componenta *Model* este alcătuită dintr-o clasă ce extinde clasa

  `com.opensymphony.xwork2.ActionSupport`

  care implementează interfaţa `Action`. Această interfaţă declară metoda

  `public String execute() throws Exception`

Programatorul ori suprascrie această metodă ori indică metoda care îndeplineşte activităţile necesare satisfacerii cererii. Această metodă se declară prin atributul `method` a elementului `action` din componenta *Controller* `struts.xml`.

În ambele cazuri se returnează un `String` fixat de clasa `ResultNames: SUCCESS, ERROR, INPUT, LOGIN, NONE`.

Parametrilor preluaţi dintr-un formular li se asociază în componenta Java câmpuri *private.* Pentru fiecare asemenea câmp se definesc metodele `set` şi / sau `get`.

- Componenta *Controller* este reprezentat de fişierul de configurare *struts.xml,* plasat în catalogul `WEB-INF\classes` al aplicaţiei Web. Prin datele acestui fişier se face asocierea dintre numele acţiunii, clasa care implementează acţiunea şi fişierele cu rezultatele prelucrărilor.

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE struts PUBLIC
  "-//Apache Software Foundation//DTD Struts Configuration 2.0//EN"
  "http://struts.apache.org/dtds/struts-2.0.dtd">
<struts>
  <package name="pachet" extends="struts-default">
     <action name="NumeActiune"
             class="pachet.Clasa"
             method="numeMetoda">
        <result>/jsp/Rezultat1.jsp</result>
        <result name="input">/jsp/Rezultat2.jsp</result>
        . . .
     </action>
        . . .
  </package>
</struts>
```

Totodată, utilizarea lui *Struts2* necesită fişierul `web.xml`

```
1 <?xml version="1.0" encoding="UTF-8"?>
2 <web-app id="WebApp_9" version="2.4"
3    xmlns="http://java.sun.com/xml/ns/j2ee"
4    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
5    xsi:schemaLocation="http://java.sun.com/xml/ns/j2ee
6    http://java.sun.com/xml/ns/j2ee/web-app_2_4.xsd">
7
8   <display-name>nume_aplicatie</display-name>
```

```
10   <filter>
11     <filter-name>struts2</filter-name>
12     <filter-class>
13       org.apache.struts2.dispatcher.ng.filter.StrutsPrepareAndExecuteFilter
14     </filter-class>
15   </filter>

17   <filter-mapping>
18     <filter-name>struts2</filter-name>
19     <url-pattern>/*</url-pattern>
20   </filter-mapping>

22   <welcome-file-list>
23     <welcome-file>index.html</welcome-file>
24   </welcome-file-list>
25 </web-app>
```

Desfăşurarea aplicaţiei Web este

```
webapps
|--> catalogul aplicatiei (context)
       |--> html
       |--> jsp
       |--> WEB-INF
       |       |--> classes
       |       |       |--> pachetul aplicatiei
       |       |       |       |    *.class
       |       |       |    struts.xml
       |       |--> lib
       |       |       *.jar
       |       |    web.xml
       |    index.html
```

1.2.2 Marcaje Struts

Marcajele Struts sunt definite într-o bibliotecă care se declară într-o pagină JSP prin

```
<%@ taglib prefix="s" uri="/struts-tags" %>
```

Nu este nevoie de specificarea lor în fişierul de configurare `web.xml`.

Amintim marcajele:

- `s:form` Pentru marcarea formularului.

  Atribute ale marcajului:

Atribut	Fel	Descriere
`action`	obligatoriu	defineşte acţiunea de executat.

- `s:textfield` Declară un câmp de introducere date de tip string.

  Atribute ale marcajului:

Atribut	Fel	Descriere
`name`	obligatoriu	Numele câmpului.
`label`	opţional	Textul explicativ al informaţiei de introdus.

- `s:password` Declară un câmp de introducere a parolei.

  Atribute ale marcajului:

Atribut	Fel	Descriere
`name`	obligatoriu	Numele câmpului.
`label`	opţional	Textul explicativ al informaţiei de introdus.

- `s:submit` Declară un buton de comandă.

  Atribute ale marcajului:

Atribut	Fel	Descriere
`value`	opţional	Textul explicativ al acţiunii

- `s:property` Afişează valoarea proprietăţii (atribut, câmp).

  Atribute ale marcajului:

Atribut	Fel	Descriere
`value`	obligatoriu	Numele proprietăţii (al câmpului)

- `s:select`

  Atribute ale marcajului:

Atribut	Fel	Descriere
`label`	opţional	Textul explicativ al controlului grafic
`name`	opţional	Numele proprietăţii
`list`	obligatoriu	Lista opţiunilor sau numele listei *Exemplu:* "{'C2F','F2C'}"
`listKey`	opţional	Valorile cheii. Proprietatea ca avea valoarea selectată.
`listValue`	opţional	Valorile afişate

- `s:hidden`

  Atribute ale marcajului:

Atribut	Fel	Descriere
`name`	obligatoriu	Numele proprietăţii / câmpului
`value`	obligatoriu	Valoarea transmisă

- `s:if`

  Atribute ale marcajului:

Atribut	Fel	Descriere
`test`	obligatoriu	Test

- `s:else`

```
<s:if test="%{incercari > 0}">
  Incercari <s:property value="%{incercari+1}"/>
</s:if>
<s:else>
  Prima incercare
</s:else>
```

Marcajele *Strut2* folosesc limbajul *Object Graph Navigation Language* (OGNL) care

- leagă elementele grafice de control de obiecte;
- crează dinamic liste şi tabele de proprietăţi pentru elementele grafice de control;
- încarcă metodele necesare rezolvării cererii clientului;
- Realizează conversia datelor din şirul de caractere recepţionat în tipurile Java (`String, boolean / Boolean, int / Integer, long / Long, float / Float, double / Double, Date, List, Map,` array).

La fiecare apelare *Struts2* dar înaintea executării solicitării cerute *struts2* crează un obiect *ValueStack* care conţine *Object Stack* - stiva obiectelor - şi *Context Map* cu atributele cererii, ale sesiunii, etc.

1.2.3 Aplicaţie Struts2 simplă

Exemplificăm prin aplicaţia simplă de calcul a celui mai mare divizor comun a două numere naturale.

Exemplul 1.2.1

Aplicaţia *Struts2* poate conţine mai multe acţiuni / activităţi. Alegerea se face dintr-o pagina *AlegeApp.html*, care ţine de partea *view* a aplicaţiei Web. Această pagină `html` reprezintă punctul de intrare in aplicaţie.

- Fişierul *struts.xml* al componentei *Controller* este

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE struts PUBLIC
  "-//Apache Software Foundation//DTD Struts Configuration 2.0//EN"
  "http://struts.apache.org/dtds/struts-2.0.dtd">
<struts>
  <package name="cmmdc" extends="struts-default">

    <action name="Cmmdc" class="cmmdc.Cmmdc"
          method="computeCmmdc">
      <result>/jsp/ResultCmmdc.jsp</result>
    </action>

   <action name="*" >
      <result>/html/AlegeApp.html</result>
    </action>

  </package>
</struts>
```

- *AlegeApp.html*

```
<html>
 <head>
   <title> Alege Applicatia </title>
 </head>
 <body bgcolor="#aaeeaa">
   <h1> Alege&#355;i aplica&#355;ia  </h1>
   <ul>
     <li>
     <a href="/mystruts2-app/jsp/Cmmdc.jsp">
      Calculul celui mai mare divizor comun a dou&#259;
      numere naturale
     </a>
     </li>
   </ul>
 </body
</html>
```

  În cazul de faţă este o singură opţiune: calculul celui mai mare divizor comun.

- Componenta *View* a acestei aplicaţii este alcătuită din două fişiere:
  - Introducerea datelor (*Cmmdc.jsp*)

```
<%@ taglib prefix="s" uri="/struts-tags" %>
<html>
  <head>
    <title>Calculul celui mai mare divizor comun a dou&#259;
       numere naturale</title>
  </head>
  <body>
    <h3> Introduce&#355;i </h3>
    <s:form action="Cmmdc.action">
       <s:textfield label="Primul numar" name="m" />
       <s:textfield label="Al doilea numar" name="n" />
       <s:submit value="Calculeaza" />
    </s:form>
  </body>
</html>
```

  - Afişarea rezultatelor (*RezultCmmdc.jsp*)

```
<%@ taglib prefix="s" uri="/struts-tags" %>
<html>
  <head>
    <title>Cmmdc</title>
  </head>
  <body>
    <h2>Cmmdc: </h2>
    <!--
        <s:property value="message" />
    -->
    <s:property value="#session.cmmdc" />
  </body>
</html>
```

- Componenta *Model* corespunzătoare acţiunii *Cmmdc* este

```
package cmmdc;
import com.opensymphony.xwork2.ActionSupport;
import com.opensymphony.xwork2.ActionContext;
import java.util.Map;

public class Cmmdc extends ActionSupport {

  public String computeCmmdc(){
    long c=cmmdc(m,n);
    // varianta de transmitere a rezultatului printr-un  camp
    //
    // setMessage((new Long(c)).toString());
    //
    // varianta de transmitere a rezultatului prin session
    Map attr=ActionContext.getContext().getSession();
    attr.put("cmmdc",(new Long(c)).toString());
    return SUCCESS;
  }

  public long cmmdc(long m, long n){. . .}
```

```
22    private long m;
23    public long getM() {
24          return m;
25      }
26      public void setM(long m) {
27          this.m = m;
28    }

30    private long n;
31    public long getN() {
32          return n;
33      }
34      public void setN(long n) {
35          this.n = n;
36    }

38    /*
39    private String message;
40    public void setMessage(String message){
41      this.message = message;
42    }
43    public String getMessage() {
44      return message;
45    }
46    */
47  }
```

Apelarea aplicaţiei se face prin *index.html*

```
1 <html>
2   <head>
3     <META HTTP-EQUIV="Refresh"
4     CONTENT="0;URL=http://localhost:8080/mystruts2-app/html/AlegeApp.html">
5   </head>
6   <body>
7     <p>Apelarea aplicatiei...</p>
8   </body>
9 </html>
```

Accesul la un obiect *session* se poate obţine prin

```
Map session=ActionContext.getContext().getSession();
session.put(key,object);
```

Validarea de bază. Clasa `ActionSupport` implementează interfaţa `com.opensymphony.xworks2.Validateable` cu metoda `public void validate()`, care permite realizarea de verificări asupra datelor încărcate.

Metoda `void addFieldError(String` *numeCamp,* `String` *mesaj*`)` afişează mesajul pe ieşirea `ResultName.INPUT`.

În cazul clasei *Cmmdc.java* completarea pentru verificarea completării câmpurilor formularului este

```
public void validate(){
   if(m==0){
     addFieldError("m","Camp necompletat");
   }
   if(n==0){
     addFieldError("n","Camp necompletat");
   }
}
```

Verificarea caracterului numeric al datelor convertite de *Struts* în tipuri numerice este făcută de OGNL.

O facilitate interesantă oferită de *Struts* este completarea automată a unei liste de opţiuni (select).

Exemplul 1.2.2 *Un fişier text conţine informaţiile {nume judeţ, capitala judeţului, abrevierea}, separate printr-un spaţiu. Se cere construirea unei aplicaţii Web care pentru un judeţ indicat, afişează informaţiile corespunzătoare din fişierul menţionat.*

Aplicaţia este alcătuită din două acţiuni:

1. Completarea listei de opţiuni în cadrul formularului. Această acţiune este lansată la apelarea aplicaţei dintr-o pagină `html`, de exemplu

```
<a href="http://localhost:8080/mystruts2-app/AlegeJudetul.action">
    Referinte despre judete
</a>
```

   - Componenta *Control*:

```
<action name="AlegeJudetul" class="appjud.ListaJudeteAction">
  <result>/jsp/AppJud.jsp</result>
</action>
```

     Pagina JSP de ieşie, *AppJud.jsp,* este pagina pe care o utilizează clientul pentru selectarea judeţului. Această pagină reprezintă componenta *View* - de apelare a acţiunii următoare.

   - Componenta *Model* este dată de clasa *ListaJudeteAction.java*

```
1  package appjud;
2  import com.opensymphony.xwork2.ActionSupport;
3  import com.opensymphony.xwork2.ActionContext;
4  import java.util.Map;
5  import java.util.List;
6  import java.util.ArrayList;
7  import java.util.HashMap;
8  import java.io.InputStream;
9  import java.io.InputStreamReader;
10 import java.io.BufferedReader;
11 import java.io.IOException;
```

```
13 public class ListaJudeteAction extends ActionSupport{
14   private HashMap<String,RefJudet> refJudete=
15     new HashMap<String,RefJudet>();

17   public List<RefJudet> getJudeteList() throws IOException{
18     List<RefJudet> list=new ArrayList<RefJudet>();
19     InputStream fis=
20       this.getClass().getResourceAsStream("judete.txt");
21     InputStreamReader isr=new InputStreamReader(fis);
22     BufferedReader br=new BufferedReader(isr);
23     String s="",jud,capit,abrev;
24     do{
25       s=br.readLine();
26       if(s!=null){
27         String[] st=s.split(" ");
28         jud=st[0];
29         capit=st[1];
30         abrev=st[2];
31         RefJudet bean=new RefJudet();
32         bean.setJud(jud);
33         bean.setCapit(capit);
34         bean.setAbrev(abrev);
35         list.add(bean);
36         refJudete.put(jud,bean);
37       }
38     }
39     while(s!=null);
40     Map attr=ActionContext.getContext().getSession();
41     attr.put("refJudete",refJudete);
42     return list;
43   }
44 }
```

unde *RefJudet.java* are codul

```
1  package appjud;
2  public class RefJudet implements java.io.Serializable{
3    private String jud;
4    private String capit;
5    private String abrev;

7    public RefJudet(){}

9    public void setJud(String jud){
10     this.jud=jud;
11   }
12   public String getJud(){
13     return jud;
14   }

16   public void setCapit(String capit){
17     this.capit=capit;
18   }
19   public String getCapit(){
20     return capit;
21   }

23   public void setAbrev(String abrev){
24     this.abrev=abrev;
```

```
25    }
26    public String getAbrev(){
27      return abrev;
28    }
29  }
```

Clasa *ListaJudeteAction* generează:

- lista *judeteList* prin metoda *getJudeteList* care va fi utilizată de *Struts* pentru completarea unei liste de opţiuni din formularul aplicaţiei. Lista este alcătuită din componente Java de tip *RefJudet.*
- un obiect de tip `HashMap<String,`*RefJudet*`>` care este reţinut de obiectul sesiune al aplicaţiei. Acest obiect conţine datele fişierului text şi este utilizat la satisfacerea cererii clientului.

2. Rezolvarea cererii clientului - adică oferirea datelor cerute pe baza selecţiei indicate.

   - Componenta *Controller*:

```
<action name="RefJudet" class="appjud.JudBean">
  <result>/jsp/ResultJud.jsp</result>
</action>
```

   - Componenta *View* (*AppJud.jsp*) este

```
1  <%@ taglib prefix="s" uri="/struts-tags" %>
2  <html>
3   <head>
4     <title>Referinte Judet</title>
5   </head>
6   <body>
7    <h1> Referinte despre judete </h1>
8    <p/>
9    <s:form
10       action="RefJudet.action">
11       <s:select name="selectat" label="Judete"
12       list="judeteList" listKey="%{jud}" listValue="%{jud}"/>
13     <s:submit/>
14   </s:form>
15 </body>
16 </html>
```

   Câmpul *selectat* va conţine numele judeţului ales. Lista *judeteList* este alcătuită din componente *RefJudet* iar *%jud* se referă ca câmpul *jud* al unei asemenea componente.

   - Componenta *View* (*ResultJud.jsp*) de afişare a rezultatelor

```
1  <%@ taglib prefix="s" uri="/struts-tags" %>
2  <html>
3    <head>
```

```
4     <title>Referite Judet</title>
5   </head>
6   <body>
7     <h2>Referintele despre judetul </h2>
8     <s:property value="jud"/>
9     <p/>
10    Capitala:
11    <s:property value="capit"/>
12    <p/>
13    Abrevierea:
14    <s:property value="abrev"/>
15  </body>
16 </html>
```

- Componenta *Model* () pentru satisfacerea cererii clientului este dată din

```
1  package appjud;
2  import java.util.Map;
3  import java.util.HashMap;
4  import com.opensymphony.xwork2.ActionSupport;
5  import com.opensymphony.xwork2.ActionContext;

7  public class JudBean extends ActionSupport{
8    private String jud=null;
9    private String capit=null;
10   private String abrev=null;
11   private String selectat;

13   public JudBean(){}

15   public String execute() throws Exception {
16     Map attr=ActionContext.getContext().getSession();
17     HashMap<String,RefJudet> refJudete=
18       (HashMap<String,RefJudet>)attr.get("refJudete");
19     RefJudet rj=refJudete.get(selectat);
20     jud=rj.getJud();
21     capit=rj.getCapit();
22     abrev=rj.getAbrev();
23     return SUCCESS;
24   }

26   public String getJud(){
27     return jud;
28   }

30   public String getCapit(){
31     return capit;
32   }

34   public String getAbrev(){
35     return abrev;
36   }

38   public void setSelectat(String selectat){
39     this.selectat=selectat;
40   }
41   public String getSelectat(){
42     return selectat;
```

```
43     }
44 }
```

Cele două părţi sunt legate prin

```
<action name="AlegeJudetul" class="appjud.ListaJudeteAction">
   <result>/jsp/AppJud.jsp</result>
</action>

<action name="RefJudet" class="appjud.JudBean">
   <result>/jsp/ResultJud.jsp</result>
</action>
```

Încărcarea unui fişier - Upload

Pentru încărcarea unui fişier, *Strut2* oferă o soluţie prefabricată, bazată pe produsul *commons-fileupload* de la *apache.*

Acţiunea - partea de control - poate fi

```
<action name="doUpload" class="upload.UploadAction" method="upload">
  <result name="success">/jsp/ResultUpload.jsp</result>
  <result name="error">/jsp/ErrorUpload.jsp</result>
</action>
```

Componenta *View* este alcătuită din

- Alegerea fişierului care se încarcă (*Upload.jsp*)

```
1  <%@ taglib prefix="s" uri="/struts-tags" %>
2  <html>
3    <head><title>Upload</title>
4    </head>
5    <body>
6      <h3> Incarcarea unui fisier (upload) </h3>
7      <s:form action="doUpload.action"
8          method="post" enctype="multipart/form-data">
9           <s:file name="myFile" label="File" />
10          <s:submit />
11     </s:form>
12   </body>
13 </html>
```

  commons-fileupload permite încărea mai multor fişiere, fapt nefolosit în pagina JSP de mai sus.

- Furnizarea unui răspuns.

La încărcarea unui fişier alături de fişierul propriu-zis sunt preluate numele şi tipul fişierului. Şablonul de prelucrare este

```
String dataDir = servletContext.getRealPath("/WEB-INF/");
File savedFile = new File(dataDir, myFileFileName);
myFile.renameTo(savedFile);
```

unde *myFile* este numele câmpului din documentul `jsp` corespunzător fişierului care se încarcă. Încărcarea se face într-o zonă de lucru al serverului Web, fiind ştearsă la finalizarea acţiunii. Este sarcina programatorului să preia fişierul *savedFile* în vederea prelucrării / salvării persistente. *myFile*`FileName` şi *myFile*`ContentType` sunt completate de *Struts* cu numele şi respectiv, tipul fişierului încărcat.

Exemplul 1.2.3 *Componenta model care preia într-un String conţinutul unui fişier text încărcat.*

Modelul este dat de clasa

```
1  package upload;
2  import org.apache.struts2.ServletActionContext;
3  import com.opensymphony.xwork2.ActionSupport;
4  import com.opensymphony.xwork2.ActionContext;
5  import java.util.Map;
6  import java.io.File;
7  import java.io.FileInputStream;
8  import java.io.InputStreamReader;
9  import java.io.BufferedReader;
10 import java.io.OutputStreamWriter;
11 import java.io.BufferedWriter;
12 import java.io.FileOutputStream;

14 public class UploadAction extends ActionSupport{
15   private File myFile;
16   private String myFileFileName;
17   private String myFileContentType;

19   public File getMyFile() {
20       return myFile;
21   }

23   public void setMyFile(File myFile) {
24       this.myFile = myFile;
25   }

27   public String getMyFileFileName() {
28       return myFileFileName;
29   }

31   public void setMyFileFileName(String myFileFileName) {
32       this.myFileFileName = myFileFileName;
33   }

35   public String getMyFileContentType() {
36       return myFileContentType;
37   }
```

```
39    public void setMyFileContentType(String myFileContentType) {
40        this.myFileContentType = myFileContentType;
41    }

43    public String upload() throws Exception {
44      Map attr=ActionContext.getContext().getSession();
45      ServletContext servletContext =
46        ServletActionContext.getServletContext();
47      if (myFile != null) {
48        String dataDir = servletContext.getRealPath("/WEB-INF/");
49        System.out.println("dataDir = "+dataDir);
50        System.out.println("FileName = "+myFileFileName);
51        File savedFile = new File(dataDir, myFileFileName);
52        myFile.renameTo(savedFile);

54        StringBuffer sb=new StringBuffer();
55        FileInputStream fis=new FileInputStream(savedFile);
56        InputStreamReader isr=new InputStreamReader(fis);
57        BufferedReader br=new BufferedReader(isr);

59        File f=new File("webapps/mystruts2-app/upload/"+myFileFileName);
60        FileOutputStream fos=new FileOutputStream(f);
61        OutputStreamWriter osw=new OutputStreamWriter(fos);
62        BufferedWriter bw=new BufferedWriter(osw);
63        String line;
64        do{
65          line=br.readLine();
66          if(line!=null){
67            sb.append(line+"\n");
68            bw.write(line,0,line.length());
69            bw.newLine();
70          }
71        }
72        while(line!=null);
73        attr.put("files",sb.toString());
74        bw.close();
75        osw.close();
76        fos.close();
77        br.close();
78        isr.close();
79        return SUCCESS;
80      }
81      else{
82        attr.put("error","Upload Error");
83        return ERROR;
84      }
85    }
86  }
```

Descărcarea unui fişier - Download

Şi pentru această problemă *Struts2* are o soluţie predefinită: este declarat un tip de răspuns `stream`, caz în care răspunsul este implicit un flux `InputStream`. În acest caz, numai pentru rezultat este nevoie de un fişier `jsp`.

Exemplul 1.2.4 *Aplicaţie Web în care clientul alege dintr-o listă un fişier*

care este descărcat.

În compunenta de control se introduce

```
<action name="doDownload" class="download.DownloadAction">
  <result type="stream">
    <param name="inputName">inputStream</param>
    <param name="bufferSize">2048</param>
    <param name="contentType">application/octet-stream</param>
  </result>
</action>
```

Lansarea aplicaţiei se face din (*Download.jsp*)

```
<%@ taglib prefix="s" uri="/struts-tags" %>
<html>
  <head><title>Download</title>
  </head>
  <body>
    <h3> Descarcarea unui fisier (download) </h3>
    <s:form
      action="doDownload.action">
      <s:select name="file" label="Alege"
        list="{'capitol.txt','xml-pic.jpg','TomJones.mp3','clock.avi'}"/>
      <s:submit value="Descarca"/>
    </s:form>
  </body>
</html>
```

Struts injectează în codul acţiunii numele fişierului de descărcat - în cazul exemplului, numele este conţinut de câmpul *file*.

Clasa modelului (acţiunii) este

```
package download;
import javax.servlet.ServletContext;
import com.opensymphony.xwork2.ActionSupport;
import com.opensymphony.xwork2.Result;
import org.apache.struts2.dispatcher.StreamResult;
import com.opensymphony.xwork2.ActionContext;
import org.apache.struts2.ServletActionContext;
import java.io.InputStream;

public class DownloadAction extends ActionSupport{
  private String file;

  public void setFile(String file) {
    this.file=file;
  }

  public InputStream getInputStream() throws Exception {
    ServletContext servletContext =
      ServletActionContext.getServletContext();
    Result result =
      ActionContext.getContext().getActionInvocation().getResult();
    // Setarea ContentDisposition are ca efect afisarea numelui
    // fisierului in fereastra de dialog de descarcare
    if (result!=null && result instanceof StreamResult){
      StreamResult streamResult = (StreamResult) result;
```

```
26          streamResult.setContentDisposition("attachment;filename="+file);
27        }
28        return servletContext.getResourceAsStream("resources/"+file);
29      }
30  }
```

1.2.4 Interceptori

Interceptorii sunt componente *Struts2* care execută sarcini înaintea sau după procesarea unei cereri. Interceptorii fixează fluxul de prelucrare al unei aplicaţii *Struts2* şi asigură realizarea de sarcini transversale (*cross-cutting tasks*).

Pachetul `struts-default` declară o familie de interceptori `defaultStack` necesară pentru îndeplinirea sarcinilor uzuale.

Exemplul 1.2.5 *Interceptorii predefiniţi* `timer` *şi* `logger` *într-o aplicaţie Struts2.*

Rezultatele sunt vizibile în fereastra serverului Web. Aplicaţia constă din: Componenta *control*

```
1  <?xml version="1.0" encoding="UTF-8" ?>
2  <!DOCTYPE struts PUBLIC
3  "-//Apache Software Foundation//DTD Struts Configuration 2.0//EN"
4  "http://struts.apache.org/dtds/struts-2.0.dtd">
5  <struts>
6    <package name="primul" extends="struts-default">
7      <action name="MyAction" class="exemplu.MyAction">
8        <interceptor-ref name="timer" />
9        <interceptor-ref name="logger" />
10       <interceptor-ref name="defaultStack" />
11       <result>success.jsp</result>
12     </action>
13   </package>
14 </struts>
```

Componenta *model*

```
1  package exemplu;
2  import com.opensymphony.xwork2.ActionSupport;

4  public class MyAction extends ActionSupport{
5    public String execute(){
6      return "success";
7    }
8  }
```

Componenta *view* conţine fişierele

- *index.jsp*

```
<%@ page contentType="text/html; charset=UTF-8"%>
<%@ taglib prefix="s" uri="/struts-tags"%>
<html>
  <head>Exemplu</head>
  <body>
    <s:form action="MyAction">
      <s:submit />
    </s:form>
  </body>
</html>
```

- *success.jsp*

```
<html>
  <body>
    Success
  </body>
</html>
```

Se pot definii interceptori proprii. Considerăm exemplul

Exemplul 1.2.6 *Aplicaţie Struts2 pentru ghicirea unui număr întreg cuprins între 1 şi 10 din 3 încercări.*

Pornind de la aplicaţia *Struts* simplă vom modifica programul prin introducerea un interceptor.

Componenta *control*

```
?xml version="1.0" encoding="UTF-8" ?>
<!DOCTYPE struts PUBLIC
  "-//Apache Software Foundation//DTD Struts Configuration 2.0//EN"
  "http://struts.apache.org/dtds/struts-2.0.dtd">

<struts>
  <package name="default" extends="struts-default">
    <action name="guess" class="exemplu.GuessNumber">
      <result name="input">/jsp/index.jsp</result>
      <result>/jsp/success.jsp</result>
      <result name="error">/jsp/error.jsp</result>
    </action>
  </package>
</struts>
```

Componenta *model*

```
package exemplu;

import java.util.Map;
import java.util.Random;
import org.apache.struts2.interceptor.SessionAware;
import com.opensymphony.xwork2.ActionSupport;

public class GuessNumber extends ActionSupport implements SessionAware{
```

```
10    private int numar;
11    private int incercari=1;
12    private Map session;
13    private int maxIncercari=4;

15    public int getNumar() {
16      return numar;
17    }
18    public void setNumar(int numar) {
19      this.numar = numar;
20    }

22    public int getIncercari() {
23      return incercari;
24    }
25    public void setIncercari(int incercari) {
26      this.incercari = incercari;
27    }

29    public void setMaxIncercari(int maxIncercari) {
30      this.maxIncercari = maxIncercari;
31    }
32    public int getMaxIncercari(){
33      return maxIncercari;
34    }

36    @Override
37    public void setSession(Map session) {
38      this.session = session;
39    }

41    public void validate() {
42      if(getNumar()==0)
43        addFieldError("numar", "Introduceti numerul");
44      if((numar>10) ||(numar<1))
45        addFieldError("numar", "Trebuie sa fie cuprins intre 1 si 10");
46    }

48    @Override
49    public String execute(){
50      int deGhicit;
51      Integer objIncercari=(Integer)session.get("incercari");
52      if(objIncercari==null){
53         Random random=new Random();
54         deGhicit=random.nextInt(10)+1;
55         session.put("guess",new Integer(deGhicit));
56      }
57      else{
58         deGhicit=((Integer)session.get("guess")).intValue();
59         incercari=objIncercari.intValue();
60      }
61      incercari++;
62      session.put("incercari",new Integer(incercari));
63      if(numar==deGhicit){
64        session.remove("guess");
65        session.remove("incercari");
66        return SUCCESS;
67      }
68      else{
```

```
69          if(incercari==maxIncercari){
70            session.remove("guess");
71            session.remove("incercari");
72            return ERROR;
73          }
74          else{
75            return INPUT;
76          }
77        }
78    }
79 }
```

Componenta *view* conţine fişierele

- *index.jsp*

```
1  <%@ page contentType="text/html; charset=UTF-8"%>
2  <%@ taglib prefix="s" uri="/struts-tags"%>
3  <html>
4    <body>
5      <h1> Ghici&#355;i num&#259;rul &#238;n 3
6          &#238;ncerc&#259;ri</h1>
7      <p/>
8      <s:if test="#session.incercari!=null">
9        &#206;ncercarea <s:property value="#session.incercari"/>
10     </s:if>
11     <s:else>
12       &#206;ncercarea 1
13     </s:else>
14     <p>
15     <s:form action="guess">
16       <s:textfield name="numar" label="Alegerea mea" />
17       <br/>
18       <s:submit value="Verifica numarul"/>
19     </s:form>
20   </body>
21 </html>
```

success.jsp

```
1  <%@ page contentType="text/html; charset=UTF-8"%>
2  <%@ taglib prefix="s" uri="/struts-tags"%>
3  <html>
4    <body>
5      <h1> Ghici&#355;i num&#259;rul &#238;n
6          <s:property value="%{maxIncercari-1}"/>&#238;ncerc&#259;ri
7      </h1>
8      <p/>
9      Num&#259;rul a fost ghicit &#238;n
10       <s:property value="%{incercari-1}"/> &#238;ncerc&#259;ri !
11     <br/>
12     <a href="http://localhost:8080/guess">
13       &#206;nc&#259; o dat&#259;
14     </a>
15   </body>
16 </html>
```

- *error.jsp*

```
<%@ page contentType="text/html; charset=UTF-8"%>
<%@ taglib prefix="s" uri="/struts-tags"%>
<html>
  <body>
    <h1> Ghici&#355;i num&#259;rul &#238;n
       <s:property value="%{maxIncercari-1}" />&#238;ncerc&#259;ri
    </h1>
    <p/>
    Ati dep&#259;&#351;it num&#259;rul de &#238;ncerc&#259;ri !
    </p>
    Num&#259;rul nu a fost ghicit!
    <br/>
    <a href="http://localhost:8080/guess">
      &#206;nc&#259; o dat&#259;
    </a>
  </body>
</html>
```

Clasa unui interceptor implementează interfaţa

`com.opensymphony.xwork2.interceptor.Interceptor.`

Interfaţa declară metodele

- `public void init()`
- `public void destroy()`
- `public String intercept(ActionInvocation actionInvocation) throws Exception`

Introducem un interceptor pentru generarea numărului ales aleator.

```

package exemplu;
import java.util.Map;
import java.util.Random;
import com.opensymphony.xwork2.Action;
import com.opensymphony.xwork2.ActionInvocation;
import com.opensymphony.xwork2.interceptor.Interceptor;

public class RandomNumberInterceptor implements Interceptor{

  @Override
  public void destroy(){
    // TODO Auto-generated method stub
  }

  @Override
  public void init(){
    // TODO Auto-generated method stub
  }

  @Override
```

```
22   public String intercept(ActionInvocation actionInvocation)
23        throws Exception{
24     // Se preia sesiunea
25     Map session=actionInvocation.getInvocationContext().getSession();
26     // Fixeaza numarul aleator la inceputul aplicatiei
27     if(!session.containsKey("guess")||!(session.get("guess") != null)){
28       Random random = new Random();
29       int deGhicit = random.nextInt(10)+1;
30       session.put("guess",deGhicit);
31     }
32     // Invocarea celorlalte sarcini
33     return actionInvocation.invoke();
34   }
35 }
```

Modificările suferite de cele trei componente sunt
Componenta *control*

```
1  <?xml version="1.0" encoding="UTF-8" ?>
2  <!DOCTYPE struts PUBLIC
3    "-//Apache Software Foundation//DTD Struts Configuration 2.0//EN"
4    "http://struts.apache.org/dtds/struts-2.0.dtd">

6  <struts>
7    <package name="default" extends="struts-default">
8      <interceptors>
9        <interceptor name="randomInterceptor"
10                    class="exemplu.RandomNumberInterceptor"/>
11       <interceptor-stack name="myStack">
12          <interceptor-ref name="defaultStack"/>
13          <interceptor-ref name="randomInterceptor"/>
14       </interceptor-stack>
15     </interceptors>
16     <action name="guess" class="exemplu.GuessNumber">
17       <interceptor-ref name="myStack"/>
18       <result name="input">/jsp/index.jsp</result>
19       <result>/jsp/success.jsp</result>
20       <result name="error">/jsp/error.jsp</result>
21     </action>
22   </package>
23 </struts>
```

Componenta *model*

```
1  package exemplu;
2  import java.util.Map;
3  import org.apache.struts2.interceptor.SessionAware;
4  import com.opensymphony.xwork2.ActionSupport;

6  public class GuessNumber extends ActionSupport
7      implements SessionAware{
8    private int numar;
9    private int incercari=1;
10   private Map session;
11   private int maxIncercari=4;

13   public int getNumar() {
14     return numar;
15   }
```

```
16    public void setNumar(int numar) {
17      this.numar = numar;
18    }

20    public int getIncercari() {
21      return incercari;
22    }
23    public void setIncercari(int incercari) {
24      this.incercari = incercari;
25    }

27    public void setMaxIncercari(int maxIncercari) {
28      this.maxIncercari = maxIncercari;
29    }
30    public int getMaxIncercari(){
31      return maxIncercari;
32    }

34    @Override
35    public void setSession(Map session) {
36      this.session = session;
37    }

39    public void validate() {
40      if(getNumar()==0) {
41        addFieldError("numar", "Introduceti numerul");
42      }
43      if((numar>10) ||(numar<1))
44        addFieldError("numar", "Trebuie sa fie cuprins intre 1 si 10");
45    }

47    @Override
48    public String execute(){
49      int deGhicit=((Integer)session.get("guess")).intValue();
50      Integer objIncercari=(Integer)session.get("incercari");
51      if(objIncercari!=null){
52         incercari=objIncercari.intValue();
53      }
54      System.out.println(incercari+" "+numar+" "+deGhicit);
55      incercari++;
56      session.put("incercari",new Integer(incercari));
57      if(numar==deGhicit){
58        session.remove("guess");
59        session.remove("incercari");
60        return SUCCESS;
61      }
62      else{
63        if(incercari==maxIncercari){
64          session.remove("guess");
65          session.remove("incercari");
66          return ERROR;
67        }
68        else{
69          return INPUT;
70        }
71      }
72    }
73  }
```

Componenta *view* nu suferă modificări.

1.2.5 Aplicaţii *Struts2* prin modelul programat

Modelul programat se obţine prin utilizarea adnotărilor. O acţiune este denumită printr-un identificator, de exemplu *xyz*. Clasa corespunzătoare va fi *actions.XyzAction.* Această clasa conţine metoda

`public String execute(),`

care este invocată de *Struts2* pentru îndeplinirea cererii clientului.

String-ul returnat fixează componenta *view* în adnotarea `org.apache.struts2.convention.annotation.Result`. Câmpurile adnotării sunt `name, location, type, params.` În cazul mai multor adnotari `Result` acestea se includ în adnotarea `org.apache.struts2.convention.annotation.Results.`

Aceste resurse se găsesc în arhiva `struts2-convention-plugin-*.jar.`

Reluăm aplicaţia din secţiunea 1.2.3.

Cmmdc.jsp:

```
<%@ taglib prefix="s" uri="/struts-tags" %>
<html>
  <head>
    <title>
      Calculul celui mai mare divizor comun a dou&#259; numere naturale
    </title>
  </head>
  <body>
    <h3> Introduce&#355;i </h3>
    <s:form action="cmmdc">
      <s:textfield label="Primul numar" name="m"/>
      <s:textfield label="Al doilea numar" name="n" />
      <s:submit value="Calculeaza"/>
    </s:form>
  </body>
</html>
```

cu acţiunea corespunzătoare

```
package actions;
import com.opensymphony.xwork2.ActionContext;
import java.util.*;
import org.apache.struts2.convention.annotation.Result;

@Result(name="success", location="/jsp/ResultCmmdc.jsp")

public class CmmdcAction{
  public String execute(){
    long c=cmmdc(m,n);
    Map attr=ActionContext.getContext().getSession();
    attr.put("cmmdc",(new Long(c)).toString());
    return "success";
  }
```

```
16    public long cmmdc(long m, long n){. . .}

18    private long m;
19    public long getM() {
20        return m;
21    }
22    public void setM(long m) {
23        this.m = m;
24    }

26    private long n;
27    public long getN() {
28        return n;
29    }
30    public void setN(long n) {
31        this.n = n;
32    }
33    private String message;
34    public void setMessage(String message){
35        this.message = message;
36    }
37    public String getMessage() {
38        return message;
39    }
40 }
```

Există o altă variantă de programare bazată pe extinderea clasei `com.opensymphony.xwork2.ActionSupport` şi bazat pe adnotarea `org.apache.struts2.convention.annotation.Action`.

```
1  package actions;
2  import com.opensymphony.xwork2.ActionSupport;
3  import org.apache.struts2.convention.annotation.Action;
4  import org.apache.struts2.convention.annotation.Result;

6  @Result(name="success", location="/jsp/ResultCmmdc.jsp")
7  @Action("cmmdc")
8  public class Cmmdc extends ActionSupport {

10     public String execute() throws Exception {
11     . . .
12 }
```

Dacă o acţiune *Struts2* produce mai multe variante de rezultat acestea se programează

```
import org.apache.struts2.convention.annotation.Result;
import org.apache.struts2.convention.annotation.Results;

@Results({
  @Result(. . .),
  @Result(. . .),
  . . .
})
```

În cazul exemplului 1.2.2 se lansează acţiunea

`http://localhost:8080/mystruts2-anapp/listajud`

cu

```
package actions;
import com.opensymphony.xwork2.ActionContext;
import java.util.*;
import java.io.*;

import org.apache.struts2.convention.annotation.Result;
@Result(name="success",location="/jsp/AppJud.jsp")

public class ListajudAction{
  private HashMap<String,RefJudet> refJudete=
    new HashMap<String,RefJudet>();

  public String execute(){
    return "success";
  }

  public List<RefJudet> getJudeteList() throws IOException{
    List<RefJudet> list=new ArrayList<RefJudet>();
    InputStream fis=this.getClass().getResourceAsStream("judete.txt");
    InputStreamReader isr=new InputStreamReader(fis);
    BufferedReader br=new BufferedReader(isr);
    String s="",jud,capit,abrev;
    do{
      s=br.readLine();
      if(s!=null){
        String[] st=s.split(" ");
        jud=st[0];
        capit=st[1];
        abrev=st[2];
        RefJudet bean=new RefJudet();
        bean.setJud(jud);
        bean.setCapit(capit);
        bean.setAbrev(abrev);
        list.add(bean);
        refJudete.put(jud,bean);
      }
    }
    while(s!=null);
    Map attr=ActionContext.getContext().getSession();
    attr.put("refJudete",refJudete);
    return list;
  }
}
```

AppJud.jsp are codul

```
<%@ taglib prefix="s" uri="/struts-tags" %>
<html>
  <head>
    <title>Referinte Judet</title>
  </head>
  <body>
    <h1> Referinte despre judete </h1>
    <p/>
    <s:form action="judbean">
      <s:select name="selectat" label="Judete"
```

```
11          list="judeteList" listKey="%{jud}" listValue="%{jud}"/>
12        <s:submit/>
13      </s:form>
14    </body>
15 </html>
```

căreie îi corespunde acţiunea dată de clasa

```
1  package actions;
2  import java.util.*;
3  import com.opensymphony.xwork2.ActionContext;
4  import org.apache.struts2.convention.annotation.Result;

6  @Result(name="success", location="/jsp/ResultJud.jsp")

8  public class JudbeanAction{
9    private String jud=null;
10   private String capit=null;
11   private String abrev=null;
12   private String selectat;

14   public String execute() throws Exception {
15     Map attr=ActionContext.getContext().getSession();
16     HashMap<String,RefJudet> refJudete =
17       (HashMap<String,RefJudet>)attr.get("refJudete");
18     RefJudet rj=refJudete.get(selectat);
19     jud=rj.getJud();
20     capit=rj.getCapit();
21     abrev=rj.getAbrev();
22     return "success";
23   }

25   public String getJud(){
26     return jud;
27   }

29   public String getCapit(){
30     return capit;
31   }

33   public String getAbrev(){
34     return abrev;
35   }

37   public void setSelectat(String selectat){
38     this.selectat=selectat;
39   }

41   public String getSelectat(){
42     return selectat;
43   }
44 }
```

Adnotările `org.apache.struts2.convention.annotation.InterceptorRef` şi `org.apache.struts2.convention.annotation.InterceptorRefs` declară interceptorii necesari unei acţiuni.

În cazul exemplului tratat, clasa interceptorului *actions.RandomNumber Interceptor* rămâne nemodificată şi interceptorul se declară în fişierul `struts.xml`

```
<?xml version="1.0" encoding="UTF-8" ?>
<!DOCTYPE struts PUBLIC
 "-//Apache Software Foundation//DTD Struts Configuration 2.0//EN"
"http://struts.apache.org/dtds/struts-2.0.dtd">
<struts>
  <package name="anint" extends="struts-default">
    <interceptors>
      <interceptor name="randomInterceptor"
       class="actions.RandomNumberInterceptor"/>
    </interceptors>
  </package>
</struts>
```

Clasa acţiunii *actions.GuessNumber* devine

```
package actions;
import java.util.Map;
import java.util.Random;
import org.apache.struts2.interceptor.SessionAware;
import com.opensymphony.xwork2.ActionSupport;
import org.apache.struts2.convention.annotation.Result;
import org.apache.struts2.convention.annotation.Results;
import org.apache.struts2.convention.annotation.Action;
import org.apache.struts2.convention.annotation.InterceptorRef;
import org.apache.struts2.convention.annotation.InterceptorRefs;
/*
@InterceptorRefs({
    @InterceptorRef("randomInterceptor"),
    @InterceptorRef("defaultStack")
})
@Action("guess")
*/
@Results({
  @Result(name="error", location="/jsp/error.jsp"),
  @Result(name="success", location="/jsp/success.jsp"),
  @Result(name="input", location="/jsp/index.jsp"),
})

@org.apache.struts2.convention.annotation.ParentPackage(value = "anint")
@Action(value="guess",
  interceptorRefs={@InterceptorRef("randomInterceptor"),
                   @InterceptorRef("defaultStack")})

public class GuessNumber extends ActionSupport implements SessionAware{
  . . .
}
```

1.2.6 *Struts 2* prin *maven*

Aplicaţiile dezvoltate corespund celor dezvoltate anterior, calculul celui mai mare divizor comun (*Cmmdc.java*) şi regăsirea datelor unui judeţ (*JudBean.java, ListaJudeteAction.java, RefJudet.java*).

Modelul descriptiv

Dezvoltarea aplicaţiei constă din:

1. Generarea aplicaţiei

```
mvn archetype:generate -B
   -DgroupId=mystruts2app
   -DartifactId=mystruts2
   -DarchetypeGroupId=org.apache.struts
   -DarchetypeArtifactId=struts2-archetype-blank
   -DarchetypeVersion=*.*.*
```

   unde *.*.* se înlocuieşte cu versiunea *Struts2* folosită.

2. Se adaptează structura de cataloage şi fişiere la

```
mystruts2
|--> src
|    |--> main
|    |    |--> java
|    |    |    |--> action
|    |    |    |    |--> cmmdc
|    |    |    |    |    |     Cmmdc.java
|    |    |    |    |--> appjud
|    |    |    |    |    |     JudBean.java
|    |    |    |    |    |     ListaJudeteAction
|    |    |    |    |    |     RefJudet.java
|    |    |--> resources
|    |    |    |    judete.txt
|    |    |    |    struts.xml
|    |    |--> webapp
|    |    |    |--> html
|    |    |    |    |   AlegeApp.html
|    |    |    |--> jsp
|    |    |    |    |   Cmmdc.jsp
|    |    |    |    |   AppJud.jsp
|    |    |    |    |   ResultCmmdc.jsp
|    |    |    |    |   ResultJud.jsp
|    |    |    |--> WEB-INF
|    |    |    |    |    web.xml
|    |    |    |    index.html
|    pom.xml
```

   În clasa *ListaJudeteAction.java* fişierul *judete.txt* se încarcă prin

```
InputStream fis=this.getClass().getResourceAsStream("../judete.txt");
```

3. Prelucrarea constă din

   (a) `mvn clean package`

   (b) Fişierul `war` care rezultă se desfăşoară în serverul Web.

Modelul programat

Indicăm doar diferenţele faţă de varianta anterioară:

1. În fişierul `pom.xml` se adaugă:

```
<dependency>
        <groupId>org.apache.struts</groupId>
        <artifactId>struts2-convention-plugin</artifactId>
        <version>${struts2.version}</version>
</dependency>
```

2. Se adaptează structura de cataloage şi fişiere la

```
mystruts2annotation
|--> src
|    |--> main
|    |    |--> java
|    |    |    |--> action
|    |    |    |    |--> cmmdc
|    |    |    |    |      |     Cmmdc.java
|    |    |    |    |--> appjud
|    |    |    |    |      |     JudBean.java
|    |    |    |    |      |     ListaJudeteAction
|    |    |    |    |      |     RefJudet.java
|    |    |--> resources
|    |    |    |    judete.txt
|    |    |--> webapp
|    |    |    |--> html
|    |    |    |    |   AlegeApp.html
|    |    |    |--> jsp
|    |    |    |    |   Cmmdc.jsp
|    |    |    |    |   AppJud.jsp
|    |    |    |    |   ResultCmmdc.jsp
|    |    |    |    |   ResultJud.jsp
|    |    |    |--> WEB-INF
|    |    |    |    |    web.xml
|    |    |    |    index.html
|    pom.xml
```

3. În paralel se adaptează pachetele claselor Java şi referinţele din fişierele `html`.

1.3 Java Server Faces

Java Server Faces (JSF) este un cadru de dezvoltare (framework) a aplicaţiilor Web, asemănător cu *Struts*.

Amintim următoarele două implementări JSF:

- JSF - RI (Reference Implementation), dezvoltat de firma Oracle-Sun Microsystems; JSF este inclusă în extensia *Java Enterprise Edition* (JEE).

  Distribuţia JSF de sine stătătoare constă dintr-un fişier `javax.faces.*.jar`, care trebuie copiat în catalogul `WEB-INF\lib` al aplicaţiei.

- MyFaces, dezvoltat în cadrul organizaţiei *apache.*

1.3.1 Structura unei aplicaţii JSF

- Partea de *view* din JSF poate fi dezvoltată utilizând:
  - *Facelets* - caz în care vorbim de pagini *Facelets,* reprezentate de fişiere cu extensia `xhtml`. Resursele necesare tehnologiei *Facelets* trebuie descărcate suplimentar.

    Resursele necesare sunt
    - `javax.faces.*.jar`
    - `javax.servlet.jsp.jstl`
  - *JSP*

    Resursele necesare sunt
    - `javax.faces.*.jar`
    - `jstl.jar, standard.jar` aflate în `TOMCAT_HOME\webapps\examples\WEB- INF\lib`. Acestea din urmă definesc *Java Standard Tag Library* (JSTL).

  În ambele cazuri se utilizează biblioteci de marcaje specifice JSF.

  Facelets se impune datorită incompatibilităţilor dintre JSF şi JSP în ciclul de activităţi pe care le desfăşoară pentru rezolvarea unei apelări.

- Partea de *model* este alcătuită din componente Java (bean) care se încarcă cu datele furnizate din paginile JSP/Facelets, asigură funcţionalitatea specifică aplicaţiei şi constituie sursa de date pentru paginile de afişare a rezultatelor. Clasele Java trebuie să implementeze interfaţa `java.io.Serializable`.

JSF instanţiază componentele Java şi injectează datele în aceste componente.

- *Controller*-ul este dat de fişierul de configurare `faces-config.xml` în care se fixează
  - legătura dintre paginile de furnizare a datelor şi cele care afişează rezultatul prelucrării, prin elementele <`navigation-rule`>. Astfel
    - elementul <`from-view-id`> conţine referinţa la pagina Facelets / JSP furnizoare de date;
    - elementul <`navigation-case`> declară o posibilitate de ieşire. Corpul acestui element conţine:
      - <`from-outcome`> element în care se declară stringul care direcţionează ieşirea.
      - <`to-view-id`> conţine referinţa la pagina Facelets / JSP de afişare a rezultatelor obţinute în urma acţiunii corespunzătoare stringului din <`from-outcome`>.

    Toate referinţa se raportatează la contextul aplicaţiei.

    Elementul <`navigation-rule`> poate fi evitat prin precizarea în instrucţiunea `return,` din codul acţiunii, a fişierului care tratează prelucrarea care urmează, de exemplu

    ```
    return "\cmmdcOutput";
    ```

  - componentele Java utilizate, prin elemente <`managed-bean`> având componenţa:
    - elementul <`managed-bean-name`> declară numele simbolic al unei componente Java (bean).
    - elementul <`managed-bean-class`> specifică clasa componentei. Referinţa se raportează la catalogul `WEB-INF\classes`.
    - elementul <`managed-bean-scope`> specifică durata de existenţă a componentei Java: `page, request, session, application`.

    Într-un element <`managed-bean`> se pot fixa valorile unor câmpuri ale componentei Java prin intermediul marcajelor

    ```
    <managed-property>
      <property-name> . . . </property-name>
      <value> . . . </value>
    </managed-property>
    ```

Elementul <managed-bean> poate fi evitat prin utilizarea adnotărilor aplicate componentei Java.

```
import javax.faces.bean.ManagedBean;
import javax.faces.bean.SessionScoped;

@ManagedBean(name="nume_componenta")
@SessionScoped  | @ApplicationScoped
@ManagedProperty(name="nume_camp",value="valoare_camp")
```

Fişierul *web.xml* este independent de aplicaţie şi în principiu are codul

```
<?xml version='1.0' encoding='UTF-8'?>
<web-app version="2.5"
    xmlns="http://java.sun.com/xml/ns/javaee"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://java.sun.com/xml/ns/javaee
    http://java.sun.com/xml/ns/javaee/web-app_2_5.xsd">

  ...
  <servlet>
    <servlet-name>Faces Servlet</servlet-name>
    <servlet-class>javax.faces.webapp.FacesServlet</servlet-class>
    <load-on-startup>1</load-on-startup>
  </servlet>

  <servlet-mapping>
    <servlet-name>Faces Servlet</servlet-name>
    <url-pattern>*.faces sau *.jsf</url-pattern>
  </servlet-mapping>
  ...
</web-app>
```

Desfăşurarea unei aplicaţii JSF este

```
|--> webapps
|     |--> app_jsf_context
|     |     |--> WEB-INF
|     |     |     |--> lib
|     |     |     |    |  javax.faces.*.jar
|     |     |     |    |  alte_arhive_jar
|     |     |     |--> classes
|     |     |     |    |--> pachetul_aplicatiei
|     |     |     |    |       *.class (componentele Java)
|     |     |     |    web.xml
|     |     |     |    faces-config.xml
|     |     |    index.html
|     |     |    pagini_JSP sau pagini_Facelets
```

Prin *index.html* se lansează aplicaţia JSF.

Un ciclu de execuţie JSF constă din

1. Afişarea componentei *view;*

2. Preluarea datelor necesare satisfacerii cererii;
3. Validarea datelor;
4. Actualizarea componentei *model* cu datele preluate;
5. Invocarea metodelor componentei *model* care rezolvă cererea clientului;
6. Afişarea răspunsului.

1.3.2 Marcaje JSF

Marcajele JSF sunt definite în două biblioteci. Ele se declară în paginile JSP prin

```
<%@ taglib uri="http://java.sun.com/jsf/html" prefix="h" %>
<%@ taglib uri="http://java.sun.com/jsf/core" prefix="f" %>
```

Nu este nevoie de specificarea lor în fişierul de configurare `web.xml`.

Amintim marcajele:

- `f:view` Vizualizează componentele grafice.
- `h:form` Pentru marcarea formularului.
- `h:outputText` Afişează un text.

  Atribute ale marcajului:

Atribut	Fel	Descriere
`value`	obligatoriu	Textul ce se afişează. Poate fi şi referinţa la câmpul unei componente Java.

- `h:inputText` Declară un câmp de introducere date de tip string.

  Atribute ale marcajului:

Atribut	Fel	Descriere
`value`	obligatoriu	Câmpul componentei Java alimentat.
`required`	opţional	`true / false`, pentru validare.
`id`	opţional	Numele simbolic al câmpului.
`validator`	opţional	Referinţa la metoda de validare.

- `h:panelGrid` Componentele incluse sunt aranjate într-un tablou.

  Atribute ale marcajului:

Atribut	Fel	Descriere
`columns`	obligatoriu	Numărul coloanelor.

- `h:selectOneMenu` Declară un control grafic de tip *select*.

  Atribute ale marcajului:

Atribut	Fel	Descriere
`value`	obligatoriu	Câmpul componentei Java alimentat.
`required`	opţional	`true / false`, pentru validare.
`id`	opţional	Numele simbolic al câmpului.
`validator`	opţional	Referinţa la metoda de validare.

  Opţiunile se definesc în elementul `<f:selectItem>` ale cărui atribute sunt

Atribut	Fel	Descriere
`itemLabel`	obligatoriu	Textul explicativ afişat.
`itemValue`	obligatoriu	Valoarea ataşată opţiunii.

  Exemplu:

```
<h:selectOneMenu  required="true" value="#{conv.tip}">
  <f:selectItem itemLabel="Celsius -> Fahrenheit" itemValue="C2F" />
  <f:selectItem itemLabel="Fahrenheit -> Celsius" itemValue="F2C" />
</h:selectOneMenu>
```

- `h:commandButton` Declară un buton de comandă.

  Atribute ale marcajului:

Atribut	Fel	Descriere
`action`	obligatoriu	Numele simbolic al acţiunii, declarat într-un marcaj `<from-outcome>` din `faces-config.xml`.
`value`	obligatoriu	Textul butonului.

- `h:messages` Afişează mesajele de eroare rezultate în urma validărilor.

- `h:message` Afişează mesajele de eroare rezultate în urma validărilor.

  Atribute ale marcajului:

Atribut	Fel	Descriere
`for`	obligatoriu	Numele simbolic al câmpului supus verificării.

1.3.3 Aplicaţii JSF cu pagini Facelets

În acest caz fişierul *web.xml* se completează este

```
<context-param>
  <param-name>javax.faces.DEFAULT_SUFFIX</param-name>
  <param-value>.xhtml</param-value>
</context-param>
```

Componenta *view* a unei aplicaţii este alcătuită din pagini *Facelets.*

Pentru început, plecăm de la codul cu marcaje JSF (*cmmdcInput.xhtml*) - corespunzătoare unui formular de date pentru calculul celui mai mare divizor comun a două numere naturale

```
1  <html>
2      <h:form>
3        <h:panelGrid columns="2">
4            <h:outputText value="Primul num&#259;r" />
5            <h:inputText value="#{cb.sm}" />
6            <h:outputText value="Al doilea num&#259;r" />
7            <h:inputText value="#{cb.sn}" />
8            <h:commandButton action="#{cb.compute}" value="Calculeaz&#259;" />
9        </h:panelGrid>
10     </h:form>
11 </html>
```

Referinţa la resursele unei componente Java se indică cuprinse prin #{ } sau ${ }. În acest cod, *cb* desemnează o componentă Java pentru calculul celui mai mare divizor comun a două numere naturale. *sm,sn* sunt câmpuri iar *compute* este o metodă - codul complet al clasei este dat mai jos.

O pagină *Facelets* poate corespunde unui şablon, o resursă *Facelets* reutilizabilă. În şablon se definesc zone a căror conţinut se poate iniţializa şi care ulterior se pot modifica. O regiune se defineşte prin

```
<ui:insert name="numeZona">continut</ui:insert>
```

Zona poate fi iniţializată cu conţinutul unui alt fişier *Facelets* sau `html`, prin

```
<ui:include src="fisier.[x]html"/>
```

Într-o pagină *Facelets* dezvoltată pe şablon, referinţa la acesta se obţine prin

```
<ui:composition template="sablon.xhtml">
  . . .
</ui:composition>
```

Conţinutul aflat înafara elementului <ui:composition> este ignorat. Acest efect este anulat dacă în schimb se utilizează

```
<ui:decorate template="sablon.xhtml">
  . . .
</ui:decorate>
```

Modificarea unei zone se programează prin

```
<ui:define name="numeZona">
  noul continut
</ui:define>
```

Introducerea / definirea unui conţinut nou *Facelets* se obţine cu

```
<ui:component>
  . . .
</ui:component>
```

Continutul aflat înafara acestui element este ignorat. Acest efect este anulat dacă în schimb se utilizează

```
<ui:fragment>
  . . .
</ui:fragment>
```

Prin elementul

```
<ui:param name="nume_parametru" value="#{componentaJava.camp}"/>
```

se pot transmite parametrii către un document `xhtml`. Utilizarea acestei variante oferă o generalitate mărită documentului `xhtml`

Fie şablonul *template.xhtml*

```
1 <?xml version='1.0' encoding='UTF-8' ?>
2 <!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
3   "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
4 <html xmlns="http://www.w3.org/1999/xhtml"
5       xmlns:ui="http://java.sun.com/jsf/facelets"
6       xmlns:f="http://java.sun.com/jsf/core"
7       xmlns:h="http://java.sun.com/jsf/html">
8   <ui:insert name="title">
```

```
9         Title
10    </ui:insert>
11    <br/>
12    <table width="100%">
13      <tr>
14        <td width="20%">
15          <ui:insert name="sidemenu">
16              Side Menu
17          </ui:insert>
18        </td>
19        <td width="80%">
20            <ui:insert name="body">
21                Body
22            </ui:insert>
23        </td>
24      </tr>
25    </table>
26    <br/>
27    <ui:insert name="subsol">
28        <ui:include src="footerTemplate.html"/>
29    </ui:insert>
30  </html>
```

unde *footerTemplate.html* are codul

```
1  <html>
2    <body>
3      <i>Facelets technology</i>
4    </body>
5  </html>
```

Între *titlu* şi *subsol* s-a introdus un tabel cu două coloane pentru un meniu (*sidemenu*) şi pentru zona *body.*

Exemplul 1.3.1 *Calculul celui mai mare divizor comun a două numere naturale.*

Pe acest şablon se dezvoltă paginile *Facelets* de introducere a datelor (*cmmdcInput.xhtml*) şi de afişare a rezultatelor (*cmmdcOutput.xhtml*):

```
1  <?xml version='1.0' encoding='UTF-8' ?>
2  <!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
3    "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
4  <html xmlns="http://www.w3.org/1999/xhtml"
5    xmlns:ui="http://java.sun.com/jsf/facelets"
6    xmlns:h="http://java.sun.com/jsf/html">

8    <ui:composition template="template.xhtml">
9      <ui:define name="title">
10       <h1>Cel mai mare divizor comun</h1>
11     </ui:define>
12     <ui:define name="body">
13       <h:form>
14         <h:panelGrid columns="2">
15           <h:outputText value="Primul num&#259;r" />
16           <h:inputText value="#{cb.sm}" />
17           <h:outputText value="Al doilea num&#259;r" />
```

```
            <h:inputText value="#{cb.sn}" />
            <h:commandButton action="#{cb.compute}" value="Calculeaz&#259;" />
          </h:panelGrid>
        </h:form>
      </ui:define>
    </ui:composition>
</html>
```

```
<?xml version='1.0' encoding='UTF-8' ?>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
    "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
   <html xmlns="http://www.w3.org/1999/xhtml"
      xmlns:ui="http://java.sun.com/jsf/facelets"
      xmlns:h="http://java.sun.com/jsf/html">

  <ui:composition template="template.xhtml">
    <ui:define name="title">
      <h1> CMMDC Rezultat </h1>
    </ui:define>
    <ui:define name="body">
      <h:outputText value="Cmmdc=" />
      <h:outputText value="#{cb.sresult}" />
    </ui:define>
  </ui:composition>
</html>
```

Componenta *model* este dată de clasa

```
package cmmdc;

public class CmmdcBean implements java.io.Serializable{
  private String sm;
  private String sn;
  private String sresult;

  public CmmdcBean(){}

  public String getSm(){
    return sm;
  }

  public void setSm(String sm){
    this.sm=sm;
  }

  public String getSn(){
    return sn;
  }

  public void setSn(String sn){
    this.sn=sn;
  }

  public String getSresult(){
    return sresult;
  }
  public String compute(){
    long m=Long.parseLong(sm);
    long n=Long.parseLong(sn);
```

```
32      long c=cmmdc(m,n);
33      sresult=(new Long(c)).toString();
34      return "OK";
35    }

37    private long cmmdc(long m,long n){. . .}
38  }
```

Componenta *controller* este dată de fişierul `faces-config.xml`

```
1  <?xml version='1.0' encoding='UTF-8'?>
2  <faces-config xmlns="http://xmlns.jcp.org/xml/ns/javaee"
3      xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
4      xsi:schemaLocation="http://xmlns.jcp.org/xml/ns/javaee
5      http://xmlns.jcp.org/xml/ns/javaee/web-facesconfig_2_2.xsd"
6      version="2.2">

8    <navigation-rule>
9      <description/>
10     <from-view-id>/cmmdcInput.xhtml</from-view-id>
11     <navigation-case>
12       <description/>
13       <from-outcome>OK</from-outcome>
14       <to-view-id>/cmmdcOutput.xhtml</to-view-id>
15     </navigation-case>
16   </navigation-rule>

18   <managed-bean>
19     <description/>
20     <managed-bean-name>cb</managed-bean-name>
21     <managed-bean-class>cmmdc.CmmdcBean</managed-bean-class>
22     <managed-bean-scope>session</managed-bean-scope>
23   </managed-bean>
24 </faces-config>
```

Aplicaţia se lansează prin (*index.html*)

```
1  <html>
2    <body>
3      <center>
4      <h1> Aplica&#355;ie JSF </h1>
5      <a href="/myjsfFacelets/cmmdcInput.faces">
6                 Aplica&#355;ie JSF
7      </a>
8      </center>
9    </body>
10 </html>
```

Extensia *faces* semnalează serverului Web că se execută o aplicaţie JSF.

Extindem aplicaţia pentru a exemplifica utilizarea unui element `<ui:component>`. În acest sens se definesc clasele *MenuItem*

```
1  public class MenuItem implements java.io.Serializable{
2    private String url;
3    private String label;

5    public void setUrl(String url){
6      this.url=url;
```

```
7    }
8    public String getUrl(){
9      return url;
10   }

12   public void setLabel(String label){
13     this.label=label;
14   }
15   public String getLabel(){
16     return label;
17   }

19   public MenuItem(){}
20   public MenuItem(String url,String label){
21     this.url=url;
22     this.label=label;
23   }
24 }
```

şi *MenuBean*

```
1  import java.util.Collection;
2  import java.util.ArrayList;
3  public class MenuBean implements java.io.Serializable{
4    private Collection<MenuItem> menus;

6    public Collection<MenuItem> getMenus(){
7      return menus;
8    }
9    public void setMenus(Collection<MenuItem> menus){
10     this.menus=menus;
11   }

13   public MenuBean(){
14     menus=new ArrayList<MenuItem>();
15     menus.add(new MenuItem("cmmdcInput.faces","CMMDC automat"));
16     menus.add(new MenuItem("cmmdcInput1.faces","CMMDC manual"));
17     menus.add(new MenuItem("appjudInput.faces","App judete"));
18   }
19 }
```

O instanţă a clasei *MenuBean* este creat de JSF prin

```
<managed-bean>
  <managed-bean-name>menuBean</managed-bean-name>
  <managed-bean-class>MenuBean</managed-bean-class>
  <managed-bean-scope>session</managed-bean-scope>
</managed-bean>
```

Acum suntem în măsură să definim componenta *view: sideMenu.xhtml* pentru elementul *sidemenu* definit în şablon.

```
1  <?xml version='1.0' encoding='UTF-8' ?>
2  <!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
3    "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
4  <html xmlns="http://www.w3.org/1999/xhtml"
5        xmlns:ui="http://java.sun.com/jsf/facelets"
6        xmlns:f="http://java.sun.com/jsf/core"
7        xmlns:h="http://java.sun.com/jsf/html"
```

```
8         xmlns:c="http://java.sun.com/jstl/core">
9   <ui:component>
10    <!--
11    <c:forEach var="menu" items="#{menuBean.menus}">
12    -->
13    <c:forEach var="menu" items="#{menus}">
14      <a href="#{menu.url}">#{menu.label}</a><br/>
15    </c:forEach>
16  </ui:component>
17 </html>
```

Linia comentată corespunde versiunii fără elementul <ui:param> a fişierului *cmmdcOutput.xhtml,* care se modifică în

```
1  <?xml version='1.0' encoding='UTF-8' ?>
2  <!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
3      "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
4  <html xmlns="http://www.w3.org/1999/xhtml"
5        xmlns:ui="http://java.sun.com/jsf/facelets"
6        xmlns:h="http://java.sun.com/jsf/html">

8    <ui:composition template="template.xhtml">
9      <ui:define name="title">
10       <h1> CMMDC Rezultat </h1>
11     </ui:define>
12     <ui:define name="sidemenu">
13       <ui:include src="sideMenu.xhtml">
14         <ui:param name="menus" value="#{menuBean.menus}"/>
15       </ui:include>
16     </ui:define>
17     <ui:define name="body">
18       <h:outputText value="Cmmdc=" />
19       <h:outputText value="#{cb.sresult}" />
20     </ui:define>
21   </ui:composition>
22 </html>
```

1.3.4 Aplicaţii JSF cu pagini JSP

Extensia `faces` a unei ancore este semnalul prin care serverul Web apelează JSF şi coincide cu declaraţia din elementul <url-pattern> din fişierul `web.xml`.

Reluăm aplicaţia dezvoltată mai sus cu pagini JSP pentru partea de *view*

Exemplul 1.3.2 *Calculul celui mai mare divizor comun a două numere naturale cu verificarea datelor.*

Completăm codul clasei *cmmdc.CmmdcBean* cu codul de verificare a datelor care se injectează *sm, sn*

```
1 package cmmdc;
2 import javax.faces.application.FacesMessage;
3 import javax.faces.component.UIComponent;
```

```
4  import javax.faces.context.FacesContext;
5  import javax.faces.validator.ValidatorException;

7  public class CmmdcBean implements java.io.Serializable{
8    . . .
9    public void validateString(FacesContext context,UIComponent component,
10     Object value)throws ValidatorException{
11     if((context==null)||(component==null)) {
12       throw new NullPointerException();
13     }
14     if(value.toString().trim().equals("")){
15       throw new ValidatorException(new FacesMessage(
16         "Nu ati completat campul"));
17     }
18     else{
19       String s=value.toString();
20       try{
21         Long.parseLong(s);
22       }
23       catch(NumberFormatException e){
24         throw new ValidatorException(new FacesMessage("Nu este numar"));
25       }
26     }
27   }
28 }
```

Partea de *view* este dată de fişierele JSP

- *apps.jsp*

```
1  <html>
2  <body>
3    <center>
4    <h1> Aplica&#355;ii JSF </h1>
5    <table border=2>
6      <tr>
7        <td>
8          <a href="/myjsfJSP/cmmdc.jsp">
9            Cmmdc cu preluare automat&#259; a datelor
10         </a>
11       </td>
12     </tr>
13   </table>
14   </center>
15 </body>
16 </html>
```

- *cmmdcInput.jsp*

```
1  <%@ taglib uri="http://java.sun.com/jsf/html" prefix="h" %>
2  <%@ taglib uri="http://java.sun.com/jsf/core" prefix="f" %>
3  <html>
4    <body>
5      <h1> Calculul Cmmdc </h1>
6      <f:view>
7        <p>
8        <h:form >
9          <h:panelGrid columns="3">
```

```
10            <h:outputText value="Primul numar este" />
11            <h:inputText id="m" value="#{cb.sm}" required="true"
12                validator="#{cb.validateString}" />
13            <h:message for="m" />
14            <h:outputText value="Al doilea numar este" />
15            <h:inputText id="n" value="#{cb.sn}" required="true"
16                validator="#{cb.validateString}" />
17            <h:message for="n" />
18            <h:commandButton id="submit" value="#{cb.compute}"
19                action="#{cb.compute}" />
20          </h:panelGrid>
21        </h:form>
22      </f:view>.
23    </body>
24 </html>
```

- *cmmdcOutput.jsp*

```
1 <%@ taglib uri="http://java.sun.com/jsf/html" prefix="h" %>
2 <%@ taglib uri="http://java.sun.com/jsf/core" prefix="f" %>
3 <html>
4   <body>
5     <f:view>
6       <h:outputText value="Cmmdc = #{cb.sresult}" />
7     </f:view>
8   </body>
9 </html>
```

Partea *controller* (`config-faces.xml`) are codul

```
1 ?xml version='1.0' encoding='UTF-8'?>

3 <faces-config xmlns="http://xmlns.jcp.org/xml/ns/javaee"
4   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
5   xsi:schemaLocation="http://xmlns.jcp.org/xml/ns/javaee
6   http://xmlns.jcp.org/xml/ns/javaee/web-facesconfig_2_2.xsd"
7   version="2.2">

9   <navigation-rule>
10    <description/>
11    <from-view-id>/cmmdcInput.jsp</from-view-id>
12    <navigation-case>
13      <description/>
14      <from-outcome>OK</from-outcome>
15      <to-view-id>/cmmdcOutput.jsp</to-view-id>
16    </navigation-case>
17  </navigation-rule>

19  <managed-bean>
20    <description/>
21    <managed-bean-name>cb</managed-bean-name>
22    <managed-bean-class>cmmdc.CmmdcBean</managed-bean-class>
23    <managed-bean-scope>session</managed-bean-scope>
24  </managed-bean>
25 </faces-config>
```

În exemplul dat, cadrul de lucru JSF injectează valorile celor două numere în componenta Java *cb.* Se pot prelua *manual* datele direct din formular prin intermediul variabilei de tip `ExternalContext,` prin care avem acces la variabile de tip `HttpServletRequest, HttpSession.`

```
ExternalContext context =
  FacesContext.getCurrentInstance().getExternalContext();
HttpServletRequest request=(HttpServletRequest)context.getRequest();
HttpSession session=(HttpSession)context.getSession(true);
```

în care caz, în fişierul *cmmdcInput.jsp* trebuie denumită formularul de preluare prin

```
<h:form id="myform" >
```

Fişierul *CmmdcBean.java* devine

```
1  package cmmdc1;
2  import javax.servlet.http.HttpServletRequest;
3  import javax.faces.context.FacesContext;
4  import javax.faces.context.ExternalContext;

6  public class CmmdcBean implements java.io.Serializable{
7    private String sresult;

9    public CmmdcBean(){}

11   private long cmmdc(long m,long n){. . .}

13   public String getSresult(){
14     return sresult;
15   }

17   public String compute(){
18     ExternalContext context =
19     FacesContext.getCurrentInstance().getExternalContext();
20     HttpServletRequest request=(HttpServletRequest)context.getRequest();
21     String sm = request.getParameter("myform:sm");
22     String sn = request.getParameter("myform:sn");
23     long m=Long.parseLong(sm);
24     long n=Long.parseLong(sn);
25     long c=cmmdc(m,n);
26     sresult=(new Long(c)).toString();
27     return "OK";
28   }
29 }
```

Componenta model a aplicaţiei de calcul a celui mai mare divizor comun a două numere, *CmmdcBean* conţine cod specific problemei dar şi cod care vizează JSF. Arhitectura pe care o vom prezenta separă cele două aspecte:

- Partea specifică, adică calculul celui mai mare divizor comun se face într-o clasă POJO (*Plain Old Java Object*) fară nici o legătură JSF, *CmmdcBean*

```
1  package cmmdc.model;
2  public class CmmdcBean implements java.io.Serializable{
3    private long m=1;
4    private long n=1;
5    private long result;

7    public CmmdcBean(){}

9    public long getM(){
10     return m;
11   }
12   public void setM(long m){
13     this.m=m;
14   }

16   public long getN(){
17     return n;
18   }
19   public void setN(long n){
20     this.n=n;
21   }

23   public long getResult(){
24     return result;
25   }

27   public void compute(){
28     result=cmmdc(m,n);
29   }

31   private long cmmdc(long m,long n){...}
32 }
```

- Aplicaţia JSF interacţionează cu clasa POJO doar prin intermediul unei clase *controller*, *CmmdcController*

```
1  ackage cmmdc.controller;
2  import javax.faces.component.UIInput;
3  import javax.faces.component.UIOutput;
4  import javax.faces.application.FacesMessage;
5  import javax.faces.context.FacesContext;
6  import cmmdc.model.CmmdcBean;

8  public class CmmdcController implements java.io.Serializable{
9    private CmmdcBean cmmdcBean;
10   private UIInput primulNumar;
11   private UIInput alDoileaNumar;
12   //private UIOutput rezultat;

14   public CmmdcBean getCmmdcBean(){
15     return cmmdcBean;
16   }
17   public void setCmmdcBean(CmmdcBean cmmdcBean){
18     this.cmmdcBean=cmmdcBean;
```

```
19    }

21    public UIInput getPrimulNumar(){
22      return primulNumar;
23    }
24    public void setPrimulNumar(UIInput primulNumar){
25      this.primulNumar=primulNumar;
26    }

28    public UIInput getAlDoileaNumar(){
29      return alDoileaNumar;
30    }
31    public void setAlDoileaNumar(UIInput alDoileaNumar){
32      this.alDoileaNumar=alDoileaNumar;
33    }

35    /*
36    public UIOutput getRezultat(){
37      return rezultat;
38    }
39    public void setRezultat(UIOutput rezultat){
40      this.rezultat=rezultat;
41    }
42    */

44    public String cmmdc(){
45      FacesContext facesContext = FacesContext.getCurrentInstance();
46      try{
47        cmmdcBean.compute();
48        //rezultat.setRendered(true);
49        facesContext.addMessage(null, new FacesMessage(
50            FacesMessage.SEVERITY_INFO, "OK", null));
51      }
52      catch(Exception e)     {
53        //rezultat.setRendered(false);
54        facesContext.addMessage(null, new FacesMessage(
55            FacesMessage.SEVERITY_ERROR, e.getMessage(), null));
56        System.out.println(e.getMessage());
57      }
58      return "OK";
59    }
60  }
```

Varianta comentată corespunde situaţiei în care afişarea rezultatului se face în pagină JSF de preluare a datelor.

Componentele *view* şi *controller* depind de locul unde are loc afişarea rezultatului:

- Afişarea se face în alt fişier JSP decât cel în care se preiau datele.
  - Componenta *view* este formată din fişierele *cmmdcInput.jsp*

```
1 <%@ taglib uri="http://java.sun.com/jsf/html" prefix="h" %>
2 <%@ taglib uri="http://java.sun.com/jsf/core" prefix="f" %>
3 <html>
```

```
4   <body>
5     <h1> Calculul Cmmdc </h1>
6     <f:view>
7       <p>
8       <h:form >
9         <h:panelGrid columns="3">
10          <h:outputText value="Primul numar este" />
11          <h:inputText id="m" required="true"
12            value="#{cmmdcController.cmmdcBean.m}"
13              binding="#{cmmdcController.primulNumar}" />
14          <h:message for="m" />
15          <h:outputText value="Al doilea numar este" />
16          <h:inputText id="n"  required="true"
17              value="#{cmmdcController.cmmdcBean.n}"
18              binding="#{cmmdcController.alDoileaNumar}"  />
19          <h:message for="n" />
20          <h:commandButton id="submit" value="Calculeaza"
21              action="#{cmmdcController.cmmdc}" />
22        </h:panelGrid>
23      </h:form>
24    </f:view>.
25  </body>
26 </html>
```

şi *cmmdcOutput.jsp*

```
1 <%@ taglib uri="http://java.sun.com/jsf/html" prefix="h" %>
2 <%@ taglib uri="http://java.sun.com/jsf/core" prefix="f" %>
3 <html>
4   <body>
5     <f:view>
6       <h:outputText value="Cmmdc = #{cmmdcController.cmmdcBean.result}" />
7     </f:view>
8   </body>
9 </html>
```

– Componenta *controller* - `faces-config.xml`

```
1 <?xml version='1.0' encoding='UTF-8'?>

3 <faces-config xmlns="http://xmlns.jcp.org/xml/ns/javaee"
4   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
5   xsi:schemaLocation="http://xmlns.jcp.org/xml/ns/javaee
6   http://xmlns.jcp.org/xml/ns/javaee/web-facesconfig_2_2.xsd"
7   version="2.2">

9   <navigation-rule>
10    <description/>
11    <from-view-id>/cmmdcInput.jsp</from-view-id>
12    <navigation-case>
13      <description/>
14      <from-outcome>OK</from-outcome>
15      <to-view-id>/cmmdcOutput.jsp</to-view-id>
16    </navigation-case>
17  </navigation-rule>

19  <managed-bean>
20    <description/>
21    <managed-bean-name>cmmdcController</managed-bean-name>
```

```
22      <managed-bean-class>
23            cmmdc.controller.CmmdcController
24      </managed-bean-class>
25      <managed-bean-scope>request</managed-bean-scope>
26      <managed-property>
27         <property-name>cmmdcBean</property-name>
28         <value>#{cmmdcBean}</value>
29      </managed-property>
30    </managed-bean>

32    <managed-bean>
33      <description/>
34      <managed-bean-name>cmmdcBean</managed-bean-name>
35      <managed-bean-class>cmmdc.model.CmmdcBean</managed-bean-class>
36      <managed-bean-scope>none</managed-bean-scope>
37    </managed-bean>
38 </faces-config>
```

- Afişarea se face în acelaşi fişier JSP în care se preiau datele.
  - Componenta *view* este formată din fişierul *cmmdc.jsp*

```
1  <%@ taglib uri="http://java.sun.com/jsf/html" prefix="h" %>
2  <%@ taglib uri="http://java.sun.com/jsf/core" prefix="f" %>
3  <html>
4    <body>
5      <h1> Calculul Cmmdc </h1>
6      <f:view>
7        <p>
8        <h:form >
9          <h:panelGrid columns="3">
10           <h:outputText value="Primul numar este" />
11           <h:inputText id="m"  required="true"
12              value="#{cmmdcController.cmmdcBean.m}"
13              binding="#{cmmdcController.primulNumar}" />
14           <h:message for="m" />
15           <h:outputText value="Al doilea numar este" />
16           <h:inputText id="n"  required="true"
17              value="#{cmmdcController.cmmdcBean.n}"
18              binding="#{cmmdcController.alDoileaNumar}"  />
19           <h:message for="n" />
20           <h:commandButton id="submit" value="Calculeaza"
21              action="#{cmmdcController.cmmdc}" />
22         </h:panelGrid>
23       </h:form>
24       <h:outputFormat binding="#{cmmdcController.rezultat}"
25              rendered="false">
26          <h:outputText
27              value="Cmmdc = #{cmmdcController.cmmdcBean.result}" />
28       </h:outputFormat>
29     </f:view>.
30   </body>
31 </html>
```

  - Componenta *controller* - `faces-config.xml`

```
1 <?xml version='1.0' encoding='UTF-8'?>
```

```
<faces-config xmlns="http://xmlns.jcp.org/xml/ns/javaee"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://xmlns.jcp.org/xml/ns/javaee
        http://xmlns.jcp.org/xml/ns/javaee/web-facesconfig_2_2.xsd"
    version="2.2">

  <managed-bean>
    <description/>
    <managed-bean-name>cmmdcController</managed-bean-name>
    <managed-bean-class>
        cmmdc.controller.CmmdcController
    </managed-bean-class>
    <managed-bean-scope>request</managed-bean-scope>
    <managed-property>
       <property-name>cmmdcBean</property-name>
       <value>#{cmmdcBean}</value>
    </managed-property>
  </managed-bean>

  <managed-bean>
    <description/>
    <managed-bean-name>cmmdcBean</managed-bean-name>
    <managed-bean-class>cmmdc.model.CmmdcBean</managed-bean-class>
    <managed-bean-scope>none</managed-bean-scope>
  </managed-bean>
</faces-config>
```

Exemplul 1.3.3 *Un fişier text conţine informaţiile {nume judeţ, capitala judeţului, abrevierea}, separate printr-un spaţiu. Se cere construirea unei aplicaţii Web care pentru un judeţ indicat, afişează informaţiile corespunzătoare din fişierul menţionat.*

Iniţializarea aplicaţiei constă dintr-o componenta Java *CountyBean* care reţine informaţiile din fişier într-o variabilă de tip `HashMap<String,`*RefJudet*`>`, unde *RefJudet.java* este

```
package appjud;
public class RefJudet implements java.io.Serializable{
  private String jud;
  private String capit;
  private String abrev;

  public RefJudet(){}

  public void setJud(String jud){
    this.jud=jud;
  }

  public String getJud(){
    return jud;
  }

  public void setCapit(String capit){
    this.capit=capit;
```

```
19    }

21    public String getCapit(){
22      return capit;
23    }

25    public void setAbrev(String abrev){
26      this.abrev=abrev;
27    }

29    public String getAbrev(){
30      return abrev;
31    }
32  }
```

Această variabilă de tip `HashMap` este reţinută de *sesiunea* aplicaţiei. Codul componentei Java *CountyBean* este

```
1  package appjud;
2  import javax.faces.model.SelectItem;
3  import java.util.ArrayList;
4  import java.util.HashMap;
5  import java.util.Collection;
6  import java.io.InputStream;
7  import java.io.InputStreamReader;
8  import java.io.BufferedReader;
9  import javax.servlet.http.HttpSession;
10 import javax.faces.context.ExternalContext;
11 import javax.faces.context.FacesContext;

13 public class CountyBean implements java.io.Serializable{
14   private ArrayList<SelectItem> judete=null;

16   private HashMap<String,RefJudet> refJudete=new HashMap<String,RefJudet>();

18   public CountyBean(){
19     judete=new ArrayList<SelectItem>(50);
20     try{
21       InputStream fis=this.getClass().getResourceAsStream("judete.txt");
22       InputStreamReader isr=new InputStreamReader(fis);
23       BufferedReader br=new BufferedReader(isr);
24       String s="",jud,capit,abrev;
25       do{
26         s=br.readLine();
27         if(s!=null){
28           String[] st=s.split(" ");
29           jud=st[0];
30           capit=st[1];
31           abrev=st[2];
32           judete.add(new SelectItem(jud,jud));

34           RefJudet rj=new RefJudet();
35           rj.setJud(jud);
36           rj.setCapit(capit);
37           rj.setAbrev(abrev);
38           refJudete.put(jud,rj);
39         }
40       }
41       while(s!=null);
```

```
42        ExternalContext context=
43          FacesContext.getCurrentInstance().getExternalContext();
44        HttpSession session=(HttpSession)context.getSession(true);
45        session.setAttribute("refJudete",refJudete);
46        br.close();
47        isr.close();
48        fis.close();
49      }
50      catch(Exception e){
51        System.out.println("CountyBeanException : "+e.getMessage());
52      }
53    }

55    public Collection<SelectItem> getJudete(){
56      return judete;
57    }
58  }
```

Clasa `SelectItem` din API-ul JSF oferă o modalitate eficientă de completare a unei componente grafice *ComboBox* dintr-un document jsp.

Componenta Java care satisface cererea clientului este *JudBean.java*

```
1  package appjud;
2  import java.util.HashMap;
3  import javax.servlet.http.HttpSession;
4  import javax.faces.context.ExternalContext;
5  import javax.faces.context.FacesContext;

7  public class JudBean implements java.io.Serializable{
8    private String jud=null;
9    private String capit=null;
10   private String abrev=null;

12   public JudBean(){}

14   public String execute(){
15     ExternalContext context =
16       FacesContext.getCurrentInstance().getExternalContext();
17     HttpSession session=(HttpSession)context.getSession(true);
18     HashMap<String,RefJudet> refJudete =
19       (HashMap<String,RefJudet>)session.getAttribute("refJudete");
20     RefJudet rj=refJudete.get(jud);
21     capit=rj.getCapit();
22     abrev=rj.getAbrev();
23     return "OK";
24   }

26   public void setJud(String jud){
27     this.jud=jud;
28   }

30   public String getJud(){
31     return jud;
32   }

34   public String getCapit(){
35     return capit;
36   }
```

```
38   public String getAbrev(){
39     return abrev;
40   }
41 }
```

Fişierele JSP ataşate aplicaţiei sunt

- *appjudInput.jsp*

```
1  <%@ taglib uri="http://java.sun.com/jsf/html" prefix="h" %>
2  <%@ taglib uri="http://java.sun.com/jsf/core" prefix="f" %>

4  <html>
5    <head>
6      <title> appjud </title>
7    </head>
8    <body>
9      <f:view>
10       <h:outputText value="Informatii despre judete" />
11       <br>
12       <h:form>
13       <h:outputText value="Referinte despre judetul : " />
14       <h:selectOneMenu  value="#{judBean.jud}" required="true">
15          <f:selectItems value="#{countyBean.judete}" />
16       </h:selectOneMenu>
17       <p>
18       <h:commandButton id="submit" value="Afiseaza"
19          action="#{judBean.execute}"/>
20       </h:form>
21     </f:view>
22   </body>
23 </html>
```

- *appjudOutput.jsp*

```
1  <%@ taglib uri="http://java.sun.com/jsf/html" prefix="h" %>
2  <%@ taglib uri="http://java.sun.com/jsf/core" prefix="f" %>
3  <html>
4    <body>
5       <f:view>
6        <h:outputText value="Referintele despre judetul " />
7        <h:outputText value="#{judBean.jud}" />
8        <p>
9        <h:outputText value="Capitala : " />
10       <h:outputText value="#{judBean.capit}" />
11       <p>
12       <h:outputText value="Abrevierea : " />
13       <h:outputText value="#{judBean.abrev}" />
14     </f:view>
15   </body>
16 </html>
```

Fişierul *config-faces.xml* se completează cu

```
<navigation-rule>
  <description/>
```

```
    <from-view-id>/appjudInput.jsp</from-view-id>
    <navigation-case>
      <description/>
      <from-outcome>OK</from-outcome>
      <to-view-id>/appjudOutput.jsp</to-view-id>
    </navigation-case>
  </navigation-rule>

  <managed-bean>
    <description/>
    <managed-bean-name>countyBean</managed-bean-name>
    <managed-bean-class>appjud.CountyBean</managed-bean-class>
    <managed-bean-scope>session</managed-bean-scope>
  </managed-bean>

  <managed-bean>
    <description/>
    <managed-bean-name>judBean</managed-bean-name>
    <managed-bean-class>appjud.JudBean</managed-bean-class>
    <managed-bean-scope>session</managed-bean-scope>
  </managed-bean>
```

Încărcarea unui fişier - Upload

O soluţie JSF pentru încărcarea unui fişier se bazează pe interfaţa `org.apache.myfaces.custom.fileupload.UploadedFile` din pachetul *tomahawk*. Şi această soluţie are la bază produsul *apache commons-fileupload.*

Pachetul *tomahawk* conţine o implementare implicită a interfaţei `UploadedFile`. Metodele declarate de această interfaţă sunt

- `byte[] getBytes()`

  Returnează conţinutul fişierului încărcat.

- `String getContentType()`

- `InputStream getInputStrean()`

  Furnizează fluxul de încărcare a fişierului.

- `String getName()`

  Furnizează numele fişierului care se încarcă.

- `long getSize()`

  Furnizează mărimea fişierului care se încarcă.

Catalogul `lib` al aplicaţiei trebuie să conţină suplimentar fişierele

commons-beanutils-*.jar
commons-el.jar
commons-collections-*.jar
commons-digester-*.jar
commons-discovery-*.jar
commons-fileupload-*.jar
commons-io-*.jar
commons-logging-*.jar
tomahawk-*.jar

Fişierul `web.xml` se completează cu

```
<filter>
  <filter-name>extensionsFilter</filter-name>
  <filter-class>
    org.apache.myfaces.webapp.filter.ExtensionsFilter
  </filter-class>
  <init-param>
    <description>Set the size limit for uploaded files.
       Format: 10 - 10 bytes 10k - 10 KB 10m - 10 MB 1g - 1 GB
    </description>
    <param-name>uploadMaxFileSize</param-name>
    <param-value>5m</param-value>
  </init-param>
  <init-param>
    <description>
       Set the threshold size - files below this limit are stored in memory,
       files above this limit are stored on disk.
       Format: 10 - 10 bytes 10k - 10 KB 10m - 10 MB 1g - 1 GB
    </description>
    <param-name>uploadThresholdSize</param-name>
    <param-value>100k</param-value>
  </init-param>
</filter>
<filter-mapping>
  <filter-name>extensionsFilter</filter-name>
  <url-pattern>*.faces</url-pattern>
</filter-mapping>
```

Componenta *control* din `faces-config.xml` este

```
  <navigation-rule>
    <from-view-id>/uploadInput.jsp</from-view-id>
    <navigation-case>
      <from-outcome>OK</from-outcome>
      <to-view-id>/uploadOKOutput.jsp</to-view-id>
    </navigation-case>
    <navigation-case>
      <from-outcome>ERROR</from-outcome>
      <to-view-id>/uploadErrorOutput.html</to-view-id>
    </navigation-case>
  </navigation-rule>
```

```
<managed-bean>
  <managed-bean-name>uploadBean</managed-bean-name>
  <managed-bean-class>upload.UploadBean</managed-bean-class>
  <managed-bean-scope>session</managed-bean-scope>
</managed-bean>
```

cu componenta *model* dat de clasa *UploadBean.java*

```
1  package upload;
2  import org.apache.myfaces.custom.fileupload.UploadedFile;
3  import java.io.InputStream;
4  import java.io.File;
5  import java.io.FileOutputStream;
6  import javax.faces.context.FacesContext;
7  import javax.servlet.ServletContext;

9  public class UploadBean implements java.io.Serializable{
10   private UploadedFile uploadedFile;
11   private String uploadedFileName;

13   public UploadedFile getUploadedFile(){
14     return uploadedFile;
15   }

17   public void setUploadedFile(UploadedFile uploadedFile) {
18     this.uploadedFile = uploadedFile;
19   }

21   public String getUploadedFileName(){
22     return uploadedFileName;
23   }

25   public String uploadFile(){
26     try {
27       FacesContext context = FacesContext.getCurrentInstance();
28       String dirPath=
29         ((ServletContext)FacesContext.getCurrentInstance().
30            getExternalContext().getContext()).
31            getRealPath("uploads");
32       InputStream is = uploadedFile.getInputStream();
33       long fileSize = uploadedFile.getSize();
34       byte [] buffer = new byte[(int)fileSize];
35       is.read(buffer,0,(int)fileSize);
36       is.close();
37       String fileName=uploadedFile.getName();
38       String fs=System.getProperty("file.separator");
39       int i=fileName.lastIndexOf(fs);
40       uploadedFileName=fileName.substring(i+1);
41       File outputFile = new File(dirPath+fs+uploadedFileName);
42       FileOutputStream fos=new FileOutputStream(outputFile);
43       fos.write(buffer);
44       fos.flush();
45       fos.close();
46       return "OK";
47     }
48     catch (Exception e) {
49       System.out.println("Upload Exception : "+e.getMessage());
50     }
51     return "ERROR";
52   }
```

```
53 }
```

Componenta *view* este dat de fişierele

- *uploadInput.jsp*

```
1 <%@ taglib uri="http://java.sun.com/jsf/core" prefix="f" %>
2 <%@ taglib uri="http://myfaces.apache.org/tomahawk" prefix="t" %>

4 <html>
5   <head>
6       <title>File upload test</title>
7   </head>
8   <body>
9     <h1> &#206;nc&#259;rcarea unui fi&#351;ier </h1>
10    <f:view>
11    <h:form id="uploadForm" enctype="multipart/form-data">
12      <div>
13        <h:outputLabel value="Alegeti fisierul" />
14        <t:inputFileUpload id="file" value="#{uploadBean.uploadedFile}"
15            required="true" />
16        <h:message for="file" style="color: red;" />
17      </div>
18      <p>
19      <div>
20        <h:commandButton value="Incarca"
21            action="#{uploadBean.uploadFile}" />
22      </div>
23      </p>
24    </h:form>
25    </f:view>
26  </body>
27 </html>
```

- *uploadOKOutput.jsp*

```
1 <%@ taglib uri="http://java.sun.com/jsf/html" prefix="h" %>
2 <%@ taglib uri="http://java.sun.com/jsf/core" prefix="f" %>

4 <html>
5   <body>
6     <center>
7     <f:view>
8     <p>
9     <h:outputText value="Fisierul #{uploadBean.uploadedFileName}
10        a fost incarcat cu succes !" />
11    </p>
12    </f:view>
13    <p>
14    <a href="uploadInput.faces"> O alt&#259; &#238;nc&#259;rcare</a>
15    </p>
16    </center>
17  </body>
18 </html>
```

- *uploadErrorOutput.jsp*

```
<html>
  <body>
    <center>
      <p>
        Inc&#259;rcare e&#351;uat&#259; !
      <p>
        <a href="uploadInput.faces">Relua&#355;i</a>
    </center>
  </body>
</html>
```

1.3.5 *JSF* dezvoltat prin *maven*

Aplicaţiile dezvoltate corespund celor dezvoltate anterior, calculul celui mai mare divizor comun (*Cmmdc.java*) şi regăsirea datelor unui judeţ (*JudBean.java, ListaJudeteAction.java, RefJudet.java*).

Dezvoltarea aplicaţiei constă din:

1. Generarea aplicaţiei

```
mvn archetype:create
   -DgroupId=myjsf
   -DartifactId=myjsfFacelets
   -DarchetypeArtifactId=maven-archetype-webapp
```

2. Se adaptează structura de cataloage şi fişiere la

```
myjsfFacelets
|--> src
|    |--> main
|    |    |--> java
|    |    |    |--> myjsf
|    |    |    |    |--> cmmdc
|    |    |    |    |      |     CmmdBean.java
|    |    |    |    |--> cmmdc1
|    |    |    |    |      |     CmmdBean.java
|    |    |    |    |--> appjud
|    |    |    |    |      |     JudBean.java
|    |    |    |    |      |     CountyBean.java
|    |    |    |    |      |     RefJudet.java
|    |    |    |--> resources
|    |    |    |    |    judete.txt
|    |    |    |--> webapp
|    |    |    |    |--> WEB-INF
|    |    |    |    |    |    web.xml
|    |    |    |    |    |    faces-config.xml
|    |    |    |    |    footerTemplate.html
|    |    |    |    |    appjudInput.xhtml
|    |    |    |    |    appjudOutput.xhtml
|    |    |    |    |    apps.xhtml
|    |    |    |    |    cmmdcInput.xhtml
|    |    |    |    |    cmmdcInput1.xhtml
```

```
|    |    |    |    cmmdcOutput.xhtml
|    |    |    |    cmmdcOutput1.xhtml
|    |    |    |    sideMenu.xhtml
|    |    |    |    template.xhtml
|    |    |    |    index.html
|    pom.xml
```

În clasa *CountyBean.java* fişierul *judete.txt* se încarcă prin

```
InputStream fis=this.getClass().getResourceAsStream("../judete.txt");
```

3. Fişierul `pom.xml` se completează cu

```
<dependency>
   <groupId>org.glassfish</groupId>
   <artifactId>javax.faces</artifactId>
   <version>*.*.*</version>
 </dependency>
 <dependency>
   <groupId>javax.servlet</groupId>
   <artifactId>servlet-api</artifactId>
   <version>2.5</version>
 </dependency>
 <dependency>
   <groupId>javax.servlet</groupId>
   <artifactId>jstl</artifactId>
   <version>1.2</version>
 </dependency>
 <dependency>
   <groupId>com.sun.facelets</groupId>
   <artifactId>jsf-facelets</artifactId>
   <version>1.1.14</version>
 </dependency>
```

unde *.*.* se înlocuieşte cu versiunea JSF folosită.

4. Prelucrarea constă din
   (a) `mvn clean package`
   (b) Fişierul `war` care rezultă se desfăşoară în serverul Web.

Capitolul 2

Asynchronous JavaScript And Xml – AJAX

AJAX - Asynchronous JavaScript And Xml - permite, pe partea de client, un schimb de date cu un program server Web, prin funcţii JavaScript. La bază se află o interfaţă de programare (API - Application Programming Interface) *XMLHttpRequest* (XHR) ce poate fi utilizară de un limbaj de scripting (JavaScript, JScript, VBScript, etc) pentru

- transfer de date către un server Web utilizând protocolul HTTP;
- manipularea datelor XML sau JSON (JavaScript Object Notation).

Caracterul asincron constă în faptul că răspunsul furnizat de un program server reface doar o parte din pagina html şi nu întrega pagina, aşa cum, de exemplu, este cazul utilizării obişnuite a unui servlet.

2.1 *AJAX* – Java

Definirea interfaţei *XMLHttpRequest* a fost intiţiată de Microsoft.
Există două implementări a interfeţei

- `ActiveXObject` în navigatorul *MS InternetExplorer;*
- `XMLHttpRequest` în celelate navigatoare.

Metodele interfeţei XMLHttpRequest.

- `open(`*method,URL*`)`
  `open(`*method,URL,async*`)`
  `open(`*method,URL,async,userName*`)`
  `open(`*method,URL,async,userName,password*`)`

  method poate fi `get` sau `post.`

  async fixează natura comunicaţiei - `true` pentru comunicaţie asincronă.

- `send(`*content*`)`

- `abord()`

- `getAllResponseHeaders()`

- `getResponseHeader(`*headerName*`)`

- `setRequestHeader(`*label,value*`)`

Proprietăţile interfeţei XMLHttpRequest.

- `onreadystatechange`

  Conţine numele funcţiei script care prelucrează răspunsul.

- `readyState`

  Indicatorul obiectului *XMLHttpRequest*: 0 - neiniţializat; 1 - deschis; 2 - trimis; 3 - recepţionat; 4 - încărcat.

- `responseText / responseXML`

  Conţine răspunsul sub forma text / xml.

- `status / statusText`

  404 - Not Found; 200 - OK.

Punctul de pornire al unei aplicaţii AJAX - Java este o pagină Web - `html.` La generarea unui eveniment legat de un element grafic al paginii Web se apelează o serie de funcţii JavaScript a căror execuţie realizează comunicaţia cu un program server - servlet sau jsp. Uzual, programul server formulează un răspuns sub forma unui fişier `xml,` care este prelucrat de o funcţie JavaScript oferind date clientului.

Simplificând, punem în evidenţă 3 funcţii JavaScript

1. Generarea unui obiect `XMLHTTPRequest`

```
function initRequest() {
  if (window.XMLHttpRequest) {
    return new XMLHttpRequest();
  }
  else
    if (window.ActiveXObject){
      return new ActiveXObject("Microsoft.XMLHTTP");
    }
}
```

2. Funcţia apelată de eveniment şi care lansează comunicaţia AJAX

```
function XXX() {
  var idField=document.getElementById("numeCimp");
  var url=
   "http://host:port/context/numeServlet?numeCimp="+
    escape(idField.value);
  var req = initRequest();
  req.onreadystatechange = function() {
    if (req.readyState == 4) {
      if (req.status == 200) {
        functiaPrelucrareRaspuns(req.responseXML);
      } else {
        alert(req.status+" : "+req.statusText);
      }
    }
  };
  req.open("GET", url, true);
  req.send(null);
}
```

Câmpul `responseXML` se utilizează pentru înmagazinarea unui răspuns în format XML, iar `responseText` se utilizează pentru preluarea unui răspuns JSON.

3. Funcţia de prelucrare a răspunsului.

Exemplul 2.1.1 *Aplicaţie Web de alegere a unei oferte. Pagina Web a aplicaţiei afişează o listă de oferte de cursuri opţionale. Un student - client - selectează cursul dorit iar selecţia este transmisă unui servlet care centralizează alegerile.*

Lista cursurilor opţionale este încărcată în momentul apelării paginii Web utilizând AJAX. Pentru AJAX, pe partea de server este un alt servlet care trimite lista cursurilor opţionale sub forma unui fişier XML.

Găzduită de *apache-tomcat* desfăşurarea aplicaţiei este

```
webapps
   |--- ajax
          |--- WEB-INF
          |      |--> classes
          |      |      |--- AJAXAlegereServlet.class
          |      |      |--- AJAXCompletareServlet.class
          |      |--> lib
          |      |      |--- gson-*.jar
          |      |--- web.xml
          |--- index.html
```

Codul Java al programului *AJAXCompletareServlet* este

```
1  import java.io.IOException;
2  import java.io.PrintWriter;
3  import javax.servlet.ServletException;
4  import javax.servlet.http.HttpServlet;
5  import javax.servlet.http.HttpServletRequest;
6  import javax.servlet.http.HttpServletResponse;
7  import com.google.gson.Gson;

9  public class AJAXCompletareServlet extends HttpServlet{
10   public void doGet(HttpServletRequest req,HttpServletResponse res)
11     throws ServletException,IOException{
12     PrintWriter out=res.getWriter();
13     String tip=req.getParameter("tip");
14     res.setHeader("Cache-Control","no-cache");
15     if(tip.equals("xml")){
16       res.setContentType("text/xml");
17       out.print("<?xml version=\"1.0\" ?>");
18       out.print("<optionale>");
19       out.print("<disciplina>");
20       out.print("<denumire> Calcul Paralel </denumire>");
21       out.print("</disciplina>");
22       out.print("<disciplina>");
23       out.print("<denumire> Tehnologii distribuite </denumire>");
24       out.print("</disciplina>");
25       out.print("<disciplina>");
26       out.print("<denumire> Rezolvarea numerica a e.d.o. </denumire>");
27       out.print("</disciplina>");
28       out.print("</optionale>");
29     }
30     else{
31       res.setContentType("application/json");
32       Disciplina an=new Disciplina("Analiza numerica");
```

```
33        Disciplina pd=new Disciplina("Programare distribuita");
34        Disciplina sm=new Disciplina("Soft matematic");
35        Disciplina[] discipline={an,pd,sm};
36        Gson gson=new Gson();
37        String json=gson.toJson(discipline);
38        out.println(json);
39      }
40      out.close();
41    }

43    public void doPost(HttpServletRequest req,HttpServletResponse res)
44      throws ServletException,IOException{
45      doGet(req,res);
46    }
47  }

49  class Disciplina{
50    private String nume;
51    Disciplina(){}

53    Disciplina(String nume){
54      this.nume=nume;
55    }
56    public String getNume(){
57      return nume;
58    }
59  }
```

Răspunsul nu se stochează la recepţie

```
response.setHeader("Cache-Control","no-cache");
```

În varianata XML, natura răspunsului este "text/xml"

```
response.setContentType("text/xml");
```

iar în varianta JSON acesta este "text/plain".

În cazul exemplului, în varianta XML răspunsul la apelarea servlet-ului este fişierul `xml`

```
1  <?xml version=\"1.0\" ?>
2  <optionale>
3    <disciplina>
4      <denumire> Calcul paralel </denumire>
5    </disciplina>
6    <disciplina>
7      <denumire> Tehnologii distribuite </denumire>
8    </disciplina>
9    <disciplina>
10     <denumire> Rezolvarea numerica a e.d.o. </denumire>
11   </disciplina>
12 </optionale>
```

iar, în varianta JSON, răspunsul este stringul

```
[{"nume":"Analiza numerica"},{"nume":"Programare distribuita"},{"nume":"Soft matematic"}]
```

Servlet-ul aplicaţiei (*AJAXalegereServlet*) este banal: confirmă clientului alegerea făcută

```
import java.io.IOException;
import java.io.PrintWriter;
import javax.servlet.ServletException;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;

public class AJAXAlegereServlet extends HttpServlet{
  public void doGet(HttpServletRequest req,HttpServletResponse res)
    throws ServletException,IOException{
    String materia=req.getParameter("alegere");
    PrintWriter out=res.getWriter();
    res.setContentType("text/html");
    out.println("<html><body>");
    out.println("<h1> Disciplina optionala selectata </h1>");
    out.println("<p>"+materia);
    out.println("</body></html>");
    out.close();
  }

  public void doPost(HttpServletRequest req,HttpServletResponse res)
    throws ServletException,IOException{
    doGet(req,res);
  }
}
```

Pagina Web de apelare a aplicaţiei (*indexXMLAlegere.html*) este

```
<html>
<head>

<script language="javascript">
<!--
function initRequest() {
  if (window.XMLHttpRequest) {
    return new XMLHttpRequest();
  }
  else if (window.ActiveXObject){
    return new ActiveXObject("Microsoft.XMLHTTP");
  }
}

function doCompletion() {
  var tipField=document.getElementById("tip");
  var url = "http://localhost:8080/ajax/completare?tip="+
    escape(tipField.value);
  var req = initRequest();
  if(req!=null){
    req.open("GET", url, true);
    req.onreadystatechange = function() {
            if (req.readyState == 4) {
                if (req.status == 200) {
                   parseMessages(req.responseXML);
                } else {
                    alert(req.status+" : "+req.statusText);
                }
```

```
29              }
30      };
31      req.send(null);
32    }
33 }

35 function parseMessages(responseXML) {
36   var optionale = responseXML.getElementsByTagName("optionale")[0];
37   var select=document.getElementById("alegere");
38   for (loop = 0; loop < optionale.childNodes.length; loop++){
39     var disciplina = optionale.childNodes[loop];
40     var denumire = disciplina.getElementsByTagName("denumire")[0];
41     var den=denumire.childNodes[0].nodeValue;
42     select.options[loop]=new Option(den,den,false,false);
43   }
44 }
45 -->
46 </script>

48 <title>
49   Auto-Completion using Asynchronous JavaScript and XML (AJAX)
50 </title>
51 </head>
52 <body onload="doCompletion()">

54  <h1>Auto-Completion using Asynchronous JavaScript and XML (AJAX)</h1>

56  <form name="autofillform"
57    action="/ajax/alegerea" method="get">


60    <b>Disciplina optional : </b>

62       <select name="alegere" id="alegere" >
63       </select>

65     <p>
66       <input type="Submit" value="Transmite">
67       <input type="reset" value="Abandon" >
68       <input type="hidden" id="tip" value="xml" >
69   </form>
70  </body>
71 </html>
```

respectiv (*indexJSONAlegere.html*)

```
1 <html>
2 <head>

4 <script language="javascript">
5 <!--
6 function initRequest() {
7   if (window.XMLHttpRequest) {
8     return new XMLHttpRequest();
9   }
10  else if (window.ActiveXObject){
11    return new ActiveXObject("Microsoft.XMLHTTP");
12  }
13 }
```

```
15 function doCompletion() {
16   var tipField=document.getElementById("tip");
17   var url = "http://localhost:8080/ajax/completare?tip="+
18     escape(tipField.value);
19   var req = initRequest();
20   if(req!=null){
21     req.open("GET", url, true);
22     req.onreadystatechange = function() {
23             if (req.readyState == 4) {
24                  if (req.status == 200) {
25                     parseMessages(req.responseText);
26                  } else {
27                     alert(req.status+" : "+req.statusText);
28                  }
29             }
30     };
31     req.send(null);
32   }
33 }

35 function parseMessages(responseText){
36   var s=eval(responseText);
37   var select=document.getElementById("alegere");
38   for(var i=0;i<s.length;i++){
39     select.options[i]=new Option(s[i].nume,s[i].nume,false,false);
40   }
41 }
42 -->
43 </script>

45 <title>
46   Auto-Completion using Asynchronous JavaScript and JSON (AJAX)
47 </title>
48 </head>
49  <body onload="doCompletion()">

51  <h1>Auto-Completion using Asynchronous JavaScript and XML (AJAX)</h1>

53  <form name="autofillform"
54    action="/ajax/alegerea" method="get">


57    <b>Disciplina optional : </b>

59       <select name="alegere" id="alegere" >
60       </select>

62     <p>
63       <input type="Submit" value="Transmite">
64       <input type="reset" value="Abandon" >
65       <input type="hidden" id="tip" value="json" >
66   </form>
67  </body>
68 </html>
```

Observaţie. În cazul utilizării programului de navigare *Mozilla Firefox* aplicaţia se va apela prin *http://host:8080/context/indexAlegere.html.*

Exemplul 2.1.2 *Calcul celui mai mare divizor comun a două numere naturale cu client AJAX.*

Programul servlet este

```
1  import java.io.IOException;
2  import java.io.PrintWriter;
3  import javax.servlet.ServletException;
4  import javax.servlet.http.HttpServlet;
5  import javax.servlet.http.HttpServletRequest;
6  import javax.servlet.http.HttpServletResponse;

8  public class AJAXCmmdcServlet extends HttpServlet{

10   public long cmmdc(long m, long n){. . .}

12   public void doGet(HttpServletRequest req,HttpServletResponse res)
13       throws ServletException,IOException{
14     String sm=req.getParameter("m"),sn=req.getParameter("n");
15     long m=Long.parseLong(sm),n=Long.parseLong(sn);
16     String tip=req.getParameter("tip");
17     long x=cmmdc(m,n);
18     PrintWriter out=res.getWriter();
19     res.setHeader("Cache-Control","no-cache");
20     if(tip.equals("xml")){
21       res.setContentType("text/xml");
22       out.print("<?xml version=\"1.0\" ?>");
23       out.print("<rezultat>");
24       out.print((new Long(x)).toString());
25       out.print("</rezultat>");
26     }
27     else{
28       res.setContentType("application/json");
29       out.println((new Long(x)).toString());
30     }
31     out.close();
32   }

34   public void doPost(HttpServletRequest req,HttpServletResponse res)
35     throws ServletException,IOException{
36     doGet(req,res);
37   }
38 }
```

Se observă diferenţa faţa de soluţia non-AJAX doar în răspunsul formulat care este un document `xml` şi nu `html`.

Clientul în format XML

```
1  <html>
2  <head>

4  <script type="text/javascript" >
5  <!--

7    function initRequest() {
8      if (window.XMLHttpRequest) {
9          return new XMLHttpRequest();
10     } else if (window.ActiveXObject){
```

```
11          return new ActiveXObject("Microsoft.XMLHTTP");
12      }
13    }

15    function compute() {
16      var mField=document.getElementById("m");
17      var nField=document.getElementById("n");
18      var tipField=document.getElementById("tip");
19      var url = "http://localhost:8080/ajax/cmmdc?m=" +
20          escape(mField.value)+"&n="+escape(nField.value) +
21          "&tip=" + escape(tipField.value);
22      var req = initRequest();
23      req.onreadystatechange = function() {
24          if (req.readyState == 4) {
25              if (req.status == 200) {
26                  parseMessages(req.responseXML);
27              } else {
28                  alert(req.status+" : "+req.statusText);
29              }
30          }
31      };
32      req.open("get", url, true);
33      req.send(null);
34    }

36    function parseMessages(responseXML) {
37      var r = responseXML.getElementsByTagName("rezultat")[0];
38      var cmmdc=r.childNodes[0].nodeValue;
39      document.getElementById("rezultat").innerHTML="Cmmdc = "+cmmdc;
40    }
41 -->
42 </script>

44 <title> Cmmdc AJAX</title>
45 </head>
46  <body>
47          <h1>Cmmdc with AJAX</h1>
48          <p>
49          Primul numar :
50          <input type="text" id="m" value="1" size="15" >
51          <p>
52          Al doilea numar :
53          <input type="text"  id="n" value="1" size="15" >
54          <input type="hidden" id="tip" value="xml" >
55          <p>
56          <input type="button" value="Calculeaza" onClick="compute()" >
57          <p>
58          Cel mai mare divizor comun a celor doua numere este
59          <p>
60          <div id="rezultat" />
61  </body>
62 </html>
```

În varianta JSON funcţia javascript de prelucrare a răspunsului este

```
function parseMessages(responseText) {
  var cmmdc=responseText;
  document.getElementById("rezultat").innerHTML="Cmmdc = "+cmmdc;
}
```

Funcţiile javascript pot fi salvate într-un fişier iar referinţa la ele se dă prin

```
<script language="javascript" src="fisier_functii.js"
</script>
```

S-au creat mai multe cadre de lucru (framework) pentru dezvoltarea aplicaţiilor Web bazate pe AJAX:

Dojo	Ext
GWT	jQuery
MooTools	OpenRico
Prototype	Scriptaculous
Yahoo User Interface Library	Backbase
Buidows	Icefaces
Isomorphic Smart Client	JackBe
Nexaweb	

Dintre acestea se remarcă prin eleganţa soluţiei *Google Web Tooltit - (GWT).*

2.2 Google Web Toolkit (GWT)

GWT permite dezvoltarea aplicaţiilor Web cu schimburi de date bazate pe protocolul *Asynchronous JavaScript And Xml - AJAX* utilizând Java. La bază protocolului AJAX se află o interfaţa de programare (API) *XMLHttpRequest* (XHR) ce poate fi utilizară de un limbaj de scripting (JavaScript, JScript, VBScript, etc) pentru

- transfer de date către un server Web utilizând protocolul HTTP;
- manipularea datelor XML sau JSON (JavaScript Object Notation).

Caracterul asincron constă în faptul că răspunsul furnizat de un program server reface doar o parte din pagina html şi nu întrega pagină, aşa cum, de exemplu, este cazul utilizării obişnuite a unui servlet.

Utilizând GWT, programatorul dezvoltă aplicaţia în Java şi html iar GWT o transformă în JavaScript. Astfel se evită programarea în JavaScript. Într-o aplicaţie GWT se dezvolta atât partea de client cât şi partea de server. Partea de server este bazată pe tehnologia servlet.

GWT este distribuit gratuit de firma Google.

Instalarea produsului constă din dezarhivarea fişierului descărcat din Internet.

2.2.1 Dezvoltarea unei aplicaţii cu GWT

O aplicaţie GWT se dezvoltă

- în linie de comandă, cu `ant,` utilizând fişierul `build.xml` generat în cadrul fiecărei aplicaţii;
- în *Eclipse,* prin folosirea unei extensii adecvate (plug-in).

În cele ce urmează se va utiliza dezvoltarea cu `ant.`

Din punct de vedere al structurii aplicaţiei GWT, acesta poate fi

- simplă, fără apel de procedură la distanţă. În acest caz, rezolvarea cererii este programată în clase aflate în catalogul `client.`
- cu apel de procedură la distanţă.

O aplicaţie GWT se iniţiază prin generarea unei structuri de cataloage şi fişiere. Dacă se doreşte realizarea unei aplicaţii cu punctul de intrare dată de clasa *context.MyApp* şi care să se afle într-un catalog *catapp,* atunci generarea se obţine prin comanda

`webAppCreator -out` *catapp context.MyApp*

lansată într-o fereastă DOS. *Context*ul poate reprezenta un şir de cataloage. Rezultatul este reprezentat în Fig. 2.1 şi corespunde unei aplicaţii de întâmpinare.

```
catapp
   |--> src
   |    |--> context
   |    |    |--> client
   |    |    |    |   MyApp.java
   |    |    |    |   GreetingService.java
   |    |    |    |   GreetingServiceAsync.java
   |    |    |--> server
   |    |    |    |   GreetingServiceImpl.java
   |    |    |    MyApp.gwt.xml
   |    |--> war
   |    |    |--> WEB-INF
   |    |    |    |--> lib
   |    |    |    |    |   gwt-servlet.jar
   |    |    |    |    web.xml
   |    |    |    MyApp.css
   |    |    |    MyApp.html
   |    .classpath
   |    .project
   |    MyApp.launch
   |    README.txt
   |    build.xml
```

Figure 2.1: Iniţializarea unei aplicaţii GWT.

Acestă structură reprezintă un proiect GWT. Spunem că aplicaţia este generată de perechea (*catapp,context.MyApp*). Proiect generat este punctul de plecare pentru construirea oricărei alte aplicaţii, a cărei dezvoltare constă în modificarea, rescrierea fişierelor create şi completarea cu altele noi. Pentru o aplicaţie GWT se mai numeşte şi modul GWT.

Fişierul *MyApp.gwt.xml* este un fişier de configurare unde trebuie declarate modulele externe utilizate.

Versiunea iniţială este

```
1  <?xml version="1.0" encoding="UTF-8"?>
2  <!--
3    When updating your version of GWT, you should also update
4    this DTD reference, so that your app can take advantage of
5    the latest GWT module capabilities.
6  -->
7  <!DOCTYPE module PUBLIC "-//Google Inc.//DTD Google Web Toolkit 2.5.1//EN"
8    "http://google-web-toolkit.googlecode.com/svn/tags/2.5.1/distro-source/
9     core/src/gwt-module.dtd">
10 <module rename-to='myapp'>
11   <!-- Inherit the core Web Toolkit stuff.                    -->
12   <inherits name='com.google.gwt.user.User'/>
```

```
14  <!-- Inherit the default GWT style sheet.  You can change       -->
15  <!-- the theme of your GWT application by uncommenting          -->
16  <!-- any one of the following lines.                            -->
17  <inherits name='com.google.gwt.user.theme.clean.Clean'/>
18  <!-- <inherits name='com.google.gwt.user.theme.standard.Standard'/> -->
19  <!-- <inherits name='com.google.gwt.user.theme.chrome.Chrome'/> -->
20  <!-- <inherits name='com.google.gwt.user.theme.dark.Dark'/>     -->

22  <!-- Other module inherits                                      -->

24  <!-- Specify the app entry point class.                         -->
25  <entry-point class='unitbv.cs.td.client.MyApp'/>

27  <!-- Specify the paths for translatable code                    -->
28  <source path='client'/>
29  <source path='shared'/>

31 </module>
```

O aplicaţie GWT poate fi executată în

- modul de dezvoltare. Rularea în acest mod se lansează prin

  ```
  ant devmode
  ```

  Verificarea aplicaţiei se face prin intermediul navigatorului implicit.

- modul Web (de producţie) - caz în care se generează arhiva `war` a aplicaţiei. Se va executa

  ```
  ant war
  ```

  Cu notaţiile utilizate mai sus, va rezulta fişierul *MyApp.war.* După desfăşurarea aplicaţiei într-un server Web, container de servlet, se va apela `http://host:  port/MyApp/MyApp.html`.

2.2.2 Aplicaţie GWT fără apel de procedură la distanţă

După generarea proiectului, aplicaţia GWT se dezvoltă parcurgând paşii (se presupune din nou că numele aplicaţiei este *MyApp* aflat în catalogul *catapp*):

1. *Proiectarea interfeţei grafice* vizează elementele care se definesc în fişierul *MyApp.html,* punctul de intrare în aplicaţie. Varianta iniţială a fişierului este

   ```
   1 <!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">
   2 <!-- The HTML 4.01 Transitional DOCTYPE declaration-->
   3 <!-- above set at the top of the file will set     -->
   4 <!-- the browser's rendering engine into           -->
   ```

```
5  <!-- "Quirks Mode". Replacing this declaration      -->
6  <!-- with a "Standards Mode" doctype is supported, -->
7  <!-- but may lead to some differences in layout.   -->

9  <html>
10   <head>
11     <meta http-equiv="content-type"
12           content="text/html; charset=UTF-8">

14     <!--                                             -->
15     <!-- Consider inlining CSS to reduce the number  -->
16     <!--      of requested files                      -->
17     <!--                                             -->
18     <link type="text/css" rel="stylesheet" href="MyApp.css">

20     <!--                                             -->
21     <!-- Any title is fine                           -->
22     <!--                                             -->
23     <title>Web Application Starter Project</title>

25     <!--                                             -->
26     <!-- This script loads your compiled module.     -->
27     <!-- If you add any GWT meta tags, they must     -->
28     <!-- be added before this line.                  -->
29     <!--                                             -->
30     <script type="text/javascript" language="javascript"
31       src="myapp/myapp.nocache.js"></script>
32   </head>

34   <!--                                               -->
35   <!-- The body can have arbitrary html, or         -->
36   <!-- you can leave the body empty if you want     -->
37   <!-- to create a completely dynamic UI.           -->
38   <!--                                               -->
39   <body>

41     <!-- OPTIONAL: include this if you want history support -->
42     <iframe src="javascript:''" id="__gwt_historyFrame" tabIndex='-1'
43       style="position:absolute;width:0;height:0;border:0">
44     </iframe>

46     <!-- RECOMMENDED if your web app will not
47          function without JavaScript enabled -->
48     <noscript>
49       <div style="width: 22em; position: absolute; left: 50%;
50         margin-left: -11em; color: red; background-color: white;
51         border: 1px solid red; padding: 4px; font-family: sans-serif">
52         Your web browser must have JavaScript enabled
53         in order for this application to display correctly.
54       </div>
55     </noscript>

57     <h1>Web Application Starter Project</h1>

59     <table align="center">
60       <tr>
61         <td colspan="2" style="font-weight:bold;">
62           Please enter your name:</td>
63       </tr>
```

```
64        <tr>
65          <td id="nameFieldContainer"></td>
66          <td id="sendButtonContainer"></td>
67        </tr>
68      </table>
69    </body>
70  </html>
```

Liniile 56–68 sunt cele care trebuie adaptate aplicaţiei dezvoltate.

Un *widget*[1] (element sau control) grafic va fi redat de navigator într-o fantă (slot) definită, uzual, printr-un container `div`

2. *Construirea interfeţei grafice* constă în definirea obiectelor Java care umplu fantele declarate mai sus. Acest lucru se programează în clasa *MyApp.java,* care implementează interfaţa `EntryPoint,` interfaţă ce declară doar metoda

   `public void onModuleLoad().`

   Implementarea acestei metode reprezintă tocmai construcţia interfeţei grafice. Interfaţa de programare GWT (API) conţine o famile de clase *widget.* O instanţa a unui *widget* se asociază fantei prin

   `RootPanel.get("`*slot*`").add(`*widget*`);`

   Dintre clasele *widget* amintim:

   - `Label`
     Constructori:
     - `Label()`
     - `Label(String` *text*`)`

     Metode:
     - `public void setText(String` *text*`)`
   - `TextBox`
     Constructori:
     - `TextBox()`

[1] *widget*=*gadget* virtual, *gadget*=dispozitiv amuzant, fără însemnătate practică.

Metode:

– `public String getText()`

- `Button`
  Constructori:
  – `Button(String` *text*`)`
  Metode:
  – `public HandlerRegistration addClickHandler(ClickHandler` *clickHandler*`)`
- Containere de *widget*

`VerticalPanel`	`VerticalSplitPanel`
`HorizontalPanel`	`HorizontalSplitPanel`
`FlowPanel`	`DockPanel`

Un *widget* se include într-un container cu metoda

`void add(Widget` *widget*`)`

3. *Generarea evenimentelor.* Activităţile / acţiunile care constituie obiectivul aplicaţiei GWT se lansează printr-un clic pe un buton. Fiecărui buton i se atribuie un obiect care implementează interfaţa `ClickHandler.` Activităţile amintite mai sus sunt definite în codul metodei `public void onClick(ClickEvent event).`

4. *Programarea activităţilor corespunzătoare evenimentelor* ataşate butoanelor, adică implementarea metodelor `onClick.`

5. *Fixarea elementelor de stil* ale elementelor grafice în fişierul *MyApp.css.* Ataşarea la un *widget* a unui element de stil se obţine cu metoda `public void addStyleName(String` *style*`).`

Urmărim aceşti paşi în

Exemplul 2.2.1 *Calculul celui mai mare divizor comun a două numere naturale.*

Se generarează proiectului GWT `unitbv.cs.td.Cmmdc`

- *Proiectarea interfeţei grafice.* Considerăm interfeţa grafică conţinută într-un container de tip `VerticalPanel` *cmmdcPanel*

CMMDC ↔ `Label` *title*
m= ↔ `Label` *mLabel*
☐ ↔ `TextBox` *mTextBox* pentru introducerea primului număr
n= ↔ `nLabel`
☐ ↔ `TextBox` *nTextBox* pentru introducerea celui de al doilea număr
Compute ↔ `Button` *button*
☐ ↔ `Label` *cmmdcLabel* pentru afişarea rezultatului

Containerul va fi redat în slotul `<div id="cmmdcPage" ></div>`.

- *Construirea interfeţei grafice.* Codul care implementează interfaţa grafică imaginată mai sus este

```
public void onModuleLoad() {
    final Label title=new Label("CMMDC");
    final Label mLabel=new Label("m=");
    final Label nLabel=new Label("n=");
    final Label cmmdcLabel=new Label();
    final TextBox mTextBox=new TextBox();
    final TextBox nTextBox=new TextBox();
    final Button button = new Button("Compute");

    VerticalPanel cmmdcPanel = new VerticalPanel();
    cmmdcPanel.add(title);
    cmmdcPanel.add(mLabel);
    cmmdcPanel.add(mTextBox);
    cmmdcPanel.add(nLabel);
    cmmdcPanel.add(nTextBox);
    cmmdcPanel.add(button);
    cmmdcPanel.add(cmmdcLabel);

    RootPanel.get("cmmdcPage").add(cmmdcPanel);
  }
}
```

- *Generarea evenimentelor.* Butonului i se asociază o instanţă a clasei *MyClickHandler,* care conţine acţiunile executate după clic pe buton.

```
MyClickHandler clickHandler=new MyClickHandler(mTextBox,nTextBox,cmmdcLabel);
button.addClickHandler(clickHandler);
```

- *Programarea activităţilor corespunzătoare evenimentelor.* Acest pas corespunde realizării clasei *MyClickHandler.* Pentru fiecare din cele două numere se verifică

  1. dacă sunt furnizare;
  2. dacă şirul de caractere introdus este număr.

  În cazul în care o condiţie de mai sus nu este îndeplinită atunci se afişează un mesaj de atenţionare, iar în caz contrar se calculează cel mai mare divizor comun.
  Codul clasei *MyClickHandler* este

```
1  class MyClickHandler implements ClickHandler{
2    TextBox mTextBox=null,nTextBox=null;
3    Label cmmdcLabel=null;

5    MyClickHandler(TextBox mTextBox,TextBox nTextBox,Label cmmdcLabel){
6      this.mTextBox=mTextBox;
7      this.nTextBox=nTextBox;
8      this.cmmdcLabel=cmmdcLabel;
9    }

11   private long cmmdc(long m,long n){. . .}

13   public void onClick(ClickEvent event){
14     String sm=mTextBox.getText();
15     String sn=nTextBox.getText();
16     long m=0,n=0;
17     if(sm.equals("")){
18       Window.alert("'m' nu este dat");
19       //GWT.log("'m' nu este dat",null);
20       cmmdcLabel.setText("?");
21       return;
22     }
23     if(sn.equals("")){
24       Window.alert("'n' nu este dat");
25       //GWT.log("'n' nu este dat",null);
26       cmmdcLabel.setText("?");
27       return;
28     }
29     try{
30       m=Long.parseLong(sm);
31     }
32     catch(NumberFormatException e){
33       Window.alert("'m' nu este numar");
34       cmmdcLabel.setText("?");
35       return;
36     }
37     try{
38       n=Long.parseLong(sn);
39     }
40     catch(NumberFormatException e){
41       Window.alert("'n' nu este numar");
42       cmmdcLabel.setText("?");
43       return;
44     }
45     long c=0;
46     if((m!=0)&&(n!=0)) c=cmmdc(m,n);
47     cmmdcLabel.setText("Cmmdc="+(new Long(c)).toString());
48   }
49 }
```

- *Fixarea elementelor de stil.* Fişierul *Cmmdc.css* conţine

```
.pc-template-btn {
  display: block;
  font-size: 16pt;
}
```

```
.label-title{
  font-family: Georgia, "Times New Roman", Times, serif;
  font-weight: bold;
  font-size: 18pt
}
```

Codul complet al clasei *Cmmdc.java* este

```
1  package unitbv.cs.td.client;
2  import com.google.gwt.event.dom.client.ClickEvent;
3  import com.google.gwt.event.dom.client.ClickHandler;
4  import com.google.gwt.core.client.EntryPoint;
5  import com.google.gwt.user.client.ui.Button;
6  import com.google.gwt.user.client.ui.DialogBox;
7  import com.google.gwt.user.client.ui.Image;
8  import com.google.gwt.user.client.ui.RootPanel;
9  import com.google.gwt.user.client.ui.VerticalPanel;
10 import com.google.gwt.user.client.ui.Widget;
11 import com.google.gwt.user.client.ui.TextBox;
12 import com.google.gwt.user.client.ui.Label;
13 import com.google.gwt.user.client.Window;

15 public class Cmmdc implements EntryPoint {
16   public void onModuleLoad() {
17     final Label title=new Label("CMMDC");
18     title.addStyleName("label-title");
19     final Label mLabel=new Label("m=");
20     final Label nLabel=new Label("n=");
21     final Label cmmdcLabel=new Label();
22     final TextBox mTextBox=new TextBox();
23     final TextBox nTextBox=new TextBox();
24     final Button button = new Button("Compute");
25     button.addStyleName("pc-template-btn");
26     VerticalPanel cmmdcPanel = new VerticalPanel();
27     cmmdcPanel.setWidth("100%");
28     cmmdcPanel.setHorizontalAlignment(VerticalPanel.ALIGN_CENTER);
29     cmmdcPanel.add(title);
30     cmmdcPanel.add(mLabel);
31     cmmdcPanel.add(mTextBox);
32     cmmdcPanel.add(nLabel);
33     cmmdcPanel.add(nTextBox);
34     MyClickHandler clickHandler=new MyClickHandler(mTextBox,nTextBox,
35       cmmdcLabel);
36     button.addClickHandler(clickHandler);
37     cmmdcPanel.add(button);
38     cmmdcPanel.add(cmmdcLabel);
39     RootPanel.get("cmmdcPage").add(cmmdcPanel);
40   }
41 }
```

2.2.3 Aplicaţie GWT cu apel de procedură la distanţă

GWT asigură posibilitatea legăturii dintre o aplicaţie dezvoltată în acest mediu şi un servlet. GWT oferă două variante de dezvoltare:

- Client al unui servlet obişnuit, caz în care vorbim de client http;

Servletul şi clientul GWT trebuie să ruleze în acelaşi server Web (*Same Origin Policy*).

- Client şi servlet dezvoltat în GWT, caz în care vorbim de apel de procedură la distanţă GWT.

  Schimburile de date au loc potrivit tehnologiei AJAX. În terminologia Google se vorbeşte de servicii implementate printr-un servlet apelat de o componentă client GWT. Servlet-ul are o construcţie specifică GWT.

  Clientul este responsabil de trimiterea cererilor şi de recepţia răspunsurilor, iar partea de server de rezolvarea cererilor. Astfel se justifică terminologia de apel de procedură la distanţă GWT. Atât clientul cât şi serverul utilizează resurse GWT.

Client HTTP

La programarea unui client HTTP sarcina programatorului constă în

1. Lansarea cererii către servlet;
2. Recepţionarea şi prelucrarea răspunsului.

Clasele implicate în comunicaţia cu servlet-ul aparţin pachetului `com.google.gwt.http.client.`

Recepţionarea şi prelucrarea răspunsului se programează implementând interfaţa `RequestCallback,` cu metodele

- `public void onError(Request` *request*`,Throwable` *exception*`)`
- `public void onResponseReceived(Request` *request*`,Response` *response*`)`

Un şablon simplu este

```
class MyRequestCallback implements RequestCallback{
  private static final int STATUS_CODE_OK=200;

  public void onError(Request request,Throwable exception){
    // tratarea erorii
  }

  public void onResponseReceived(Request request,Response response){
    int sc=response.getStatusCode();
```

```
    if(sc==STATUS_CODE_OK){
      // prelucrarea raspunsului
    }
    else{
      // tratarea mesajului receptionat de la server
    }
  }
}
```

Şablonul de lansare a unei cereri GET către servlet este

```
public void doGet(String url){
  RequestBuilder rb=new RequestBuilder(RequestBuilder.GET,url);
  try{
    Request response=rb.sendRequest(null, RequestCallback);
  }
  catch(RequestException e){
    // tratarea exceptiei
  }
}
```

Şablonul de lansare a unei cereri POST către servlet este

```
public void doPost(String url,String postData){
  RequestBuilder rb=new RequestBuilder(RequestBuilder.POST,url);
  rb.setHeader("Content-Type","application/x-www-form-urlencoded");
  try{
    Request response=rb.sendRequest(postData, RequestCallback);
  }
  catch(RequestException e){
    // tratarea exceptiei
  }
}
```

Pregătirea datelor în vederea expedierii se poate programa prin

```
StringBuffer sb=new StringBuffer();
String encodedParam=URL.encodeComponent(paramName)+"="+
  URL.encodeComponent(paramValue);
sb.append(encodedM);
sb.append("&");
. . .
String postData=sb.toString();
```

O aplicaţie client http se poate verifica doar în modul Web.

Exemplul 2.2.2 *Client http dezvoltat pentru servlet-ul CmmdcServlet. Interfaţa grafică este identică cu cea din aplicaţia anterioară, de calcul a celui mai mare divizor comun a două numere naturale (Cmmdc.java).*

Metodele `onModuleLoad` ale claselor *HttpClient* şi *Cmmdc* coincid. Reamintim că servlet-ul *CmmdcServlet* preia 3 parametrii: cele două numere (m, n) ale căror cel mai mare divizor comun se calculează şi un parametru *tip* care precizează natura raspunsului ("text/html" sau "text/plain").

Apelarea servlet-ului este făcută în metoda `onClick()` a clasei ce implementează interfaţa `ClickHandler,` în cazul de faţă *MyClickHandler.*

Codul clientului este

```
1  package unitbv.cs.td.client;
2  import com.google.gwt.core.client.EntryPoint;
3  import com.google.gwt.user.client.ui.*;
4  import com.google.gwt.http.client.*;
5  import com.google.gwt.user.client.Window;
6  import com.google.gwt.event.dom.client.*;

8  public class HttpClient implements EntryPoint {
9    public void onModuleLoad() {
10     Label title=new Label("CMMDC");
11     title.addStyleName("label-title");
12     Label mLabel=new Label("m=");
13     Label nLabel=new Label("n=");
14     Label cmmdcLabel=new Label();
15     TextBox mTextBox=new TextBox();
16     TextBox nTextBox=new TextBox();
17     Button button = new Button("Compute");
18     button.addStyleName("pc-template-btn");
19     VerticalPanel cmmdcPanel = new VerticalPanel();
20     cmmdcPanel.setWidth("100%");
21     cmmdcPanel.setHorizontalAlignment(VerticalPanel.ALIGN_CENTER);
22     cmmdcPanel.add(title);
23     cmmdcPanel.add(mLabel);
24     cmmdcPanel.add(mTextBox);
25     cmmdcPanel.add(nLabel);
26     cmmdcPanel.add(nTextBox);
27     MyClickHandler clickHandler=
28       new MyClickHandler(mTextBox,nTextBox,cmmdcLabel);
29     button.addClickHandler(clickHandler);
30     cmmdcPanel.add(button);
31     cmmdcPanel.add(cmmdcLabel);
32     RootPanel.get("cmmdcPage").add(cmmdcPanel);
33   }
34 }

36 class MyClickHandler implements ClickHandler{
37   TextBox mTextBox=null,nTextBox=null;
38   Label cmmdcLabel=null;

40   MyClickHandler(TextBox mTextBox,TextBox nTextBox,Label cmmdcLabel){
```

```
41      this.mTextBox=mTextBox;
42      this.nTextBox=nTextBox;
43      this.cmmdcLabel=cmmdcLabel;
44    }

46    public void doGet(String url){
47      RequestBuilder rb=new RequestBuilder(RequestBuilder.GET,url);
48      MyRequestCallback rc=new MyRequestCallback(cmmdcLabel);
49      try{
50        Request response=rb.sendRequest(null,rc);
51      }
52      catch(RequestException e){
53        cmmdcLabel.setText("RequestException : "+e.getMessage());
54      }
55    }

57    public void onClick(ClickEvent event) {
58      String url="/appcmmdc/cmmdc";
59      String sm=mTextBox.getText();
60      String sn=nTextBox.getText();
61      long m=0,n=0;
62      if(sm.equals("")){
63        Window.alert("\'m\' nu este dat");
64        cmmdcLabel.setText("?");
65        return;
66      }
67      if(sn.equals("")){
68        Window.alert("\'n\' nu este dat");
69        cmmdcLabel.setText("?");
70        return;
71      }
72      try{
73        m=Long.parseLong(sm);
74      }
75      catch(NumberFormatException e){
76        Window.alert("\'m\' nu este numar");
77        cmmdcLabel.setText("?");
78        return;
79      }
80      try{
81        n=Long.parseLong(sn);
82      }
83      catch(NumberFormatException e){
84        Window.alert("\'n\' nu este numar");
85        cmmdcLabel.setText("?");
86        return;
87      }
88      if((m!=0)&&(n!=0)){
89        String urlExtins=url+"?m="+sm+"&n="+sn+"&tip=text/plain";
90        doGet(urlExtins);
91      }
92    }
93  }

95  class MyRequestCallback implements RequestCallback{
96    Label label;
97    private static final int STATUS_CODE_OK=200;

99    MyRequestCallback(Label label){
```

```
100     this.label=label;
101   }

103   public void onError(Request request,Throwable e){
104     label.setText("Connection error : "+e.getMessage());
105   }

107   public void onResponseReceived(Request request,Response response){
108     int sc=response.getStatusCode();
109     if(sc==STATUS_CODE_OK){
110       label.setText(response.getText());
111     }
112     else{
113       label.setText("STATUS CODE : "+sc);
114     }
115   }
116 }
```

Apel de procedură la distanţă GWT

Construcţia clientului şi a serverului este supusă unor restricţii: metodele serviciului sunt definite într-o interfaţă care este implementată de server în clase POJO.

Dezvoltarea clientului

Partea de client este alcătuită din:

1. *Interfaţa serviciului.*

   Metodele care pot fi invocate de client se declară într-o interfaţă

   ```
   package ...client;
   import com.google.gwt.user.client.rpc.RemoteService;

   public interface InterfataService extends RemoteService{
      public tip metoda1(tip var,. . .);
      public tip metoda2(tip var,. . .);
      . . .
   }
   ```

2. *Interfaţa asincronă.*

   Interfaţa asincronă are rolul de a semnala clientului recepţionarea răspunsului dat de servlet. Numele acestei interfeţe se formează din numele interfeţei serviciului plus sufixul `Async`. Cele două interfeţe conţin aceleaşi metode dar au semnături diferite.

```
package ...client;
import com.google.gwt.user.client.rpc.AsyncCallback;

public interface InterfataServiceAsync{
   public void metoda1(tip var,. . .,AsyncCallback<tip> callback);
   public void metoda2(tip var,. . .,AsyncCallback<tip> callback);
   . . .
}
```

tip corespunde clasei obiectului returnat de metodă.

Dacă o metodă returnează o valoare de tip predefinit atunci *tip* se ia clasa acoperitoare.

Dacă metoda reîntoarce `void` atunci *tip*=`Void`.

3. Interfaţă asincronă este implementată de client şi conţine prelucrările referitoare la recepţia răspunsului furnizat de server. Interfaţa `AsyncCallback` conţine două metode

   - `public void onSuccess(Object result)`
   - `public void onFailure(Throwable caught)`

   Programatorul defineşte acţiunile care se fac în cazul în care apelul metodei la distanţă s-a terminat cu succes, respectiv cu insucces.

   Structura clasei este

```
package ...client;

import com.google.gwt.user.client.rpc.AsyncCallback;
import com.google.gwt.core.client.GWT;
import com.google.gwt.user.client.Window;
import com.google.gwt.user.client.ui.*;

public class ClientCallback implements AsyncCallback {
  public void onFailure(Throwable caught) {
    GWT.log("Error ", caught);
    caught.printStackTrace();
    Window.alert(caught.toString());
  }
```

```
  public void onSuccess(Object result) {
    // cod client
    . . .
  }
}
```

4. Clientul propriu-zis este o aplicaţie GWT

```
package ...client;

import com.google.gwt.core.client.EntryPoint;
import com.google.gwt.core.client.GWT;
import com.google.gwt.user.client.ui.*;
import com.google.gwt.user.client.rpc.ServiceDefTarget;

public class Client implements EntryPoint {

  public void apelMetoda1(. . .){
    InterfataServiceAsync myService=
      (InterfataServiceAsync)GWT.create(InterfataService.class);
    ServiceDefTarget target=(ServiceDefTarget)myService;
    String relativeURL=GWT.getModuleBaseURL()+"urlPattern";
    target.setServiceEntryPoint(relativeURL);
    myService.metoda1(variabile actuale pentre metoda1,
       new ClientCallback());
  }

  public void onModuleLoad(){
    . . .
    final Button button=new Button(. . .);
    . . .

    button.addClickHandler(new ClickHandler(){
      public void onClick(ClickEvent event){
        . . .
        apelMetoda1(lista variabile);
        }
      }
    });
```

```
        . . .
        RootPanel.get("slot").add(button);
      }
    }
```

unde *slot* corespunde containerului `div` în care se include butonul.

Dezvoltarea serverului

Programul server implementează metodele interfeţei declarată în pachetul client.

```
package ...server;
import client.*;
import com.google.gwt.user.server.rpc.RemoteServiceServlet;

public class ServiceImpl extends RemoteServiceServlet
      implements InterfataService{
   public tip metoda1(tip var,. . .){. . .}
   public tip metoda2(tip var,. . .){. . .}
   . . .
}
```

Legătura dintre client şi server se face prin intermediul fişierului `web.xml` prin adăugarea elementelor

```
<servlet>
  <servlet-name>nume-servlet</servlet-name>
  <servlet-class>. . .server.ServiceImpl</servlet-class>
</servlet>

<servlet-mapping>
  <servlet-name>nume-servlet</servlet-name>
  <url-pattern>/myapp/urlPattern</url-pattern>
</servlet-mapping>
```

Exemplul 2.2.3 *Transformăm programul GWT de calcul a celui mai mare divizor comun a două numere naturale într-o aplicaţie cu apel la distanţă GWT.*

Interfaţa aplicaţiei este

```
1 package unitbv.cs.td.client;
2 import com.google.gwt.user.client.rpc.RemoteService;

4 public interface CmmdcService extends RemoteService{
5   public long cmmdc(long m,long n);
6 }
```

iar interfaţa asincronă

```
1  package unitbv.cs.td.client;
2  import com.google.gwt.user.client.rpc.AsyncCallback;

4  public interface CmmdcServiceAsync{
5    public void cmmdc(long m,long n,AsyncCallback<Long> callback);
6  }
```

Implementarea interfeţei asincrone este

```
1  package unitbv.cs.td.client;

3  import com.google.gwt.user.client.rpc.AsyncCallback;
4  import com.google.gwt.core.client.GWT;
5  import com.google.gwt.user.client.Window;

7  public class CmmdcCallback implements AsyncCallback {
8    public void onFailure(Throwable caught) {
9      GWT.log("Error ", caught);
10     caught.printStackTrace();
11     Window.alert(caught.toString());
12   }
13   public void onSuccess(Object result) {
14     long rez=((Long)result).longValue();
15     Window.alert("Cmmdc="+rez);
16   }
17 }
```

cu clientul

```
1  import com.google.gwt.core.client.EntryPoint;
2  import com.google.gwt.user.client.ui.Button;
3  import com.google.gwt.user.client.ui.DialogBox;
4  import com.google.gwt.user.client.ui.Image;
5  import com.google.gwt.user.client.ui.RootPanel;
6  import com.google.gwt.user.client.ui.VerticalPanel;
7  import com.google.gwt.user.client.ui.Widget;
8  import com.google.gwt.user.client.ui.TextBox;
9  import com.google.gwt.user.client.ui.Label;
10 import com.google.gwt.user.client.Window;
11 import com.google.gwt.core.client.GWT;
12 import com.google.gwt.user.client.rpc.ServiceDefTarget;
13 import com.google.gwt.event.dom.client.ClickEvent;
14 import com.google.gwt.event.dom.client.ClickHandler;

16 public class CmmdcClient implements EntryPoint {
17   public void getResult(long m,long n){
18     CmmdcServiceAsync cmmdcService=
19       (CmmdcServiceAsync)GWT.create(CmmdcService.class);
20     ServiceDefTarget sdt=(ServiceDefTarget)cmmdcService;
21     String endpoint=GWT.getModuleBaseURL()+"cmmdcrpc";
22     sdt.setServiceEntryPoint(endpoint);
23     cmmdcService.cmmdc(m,n,new CmmdcCallback());
24   }

26   public void onModuleLoad() {
27     Label title=new Label("CMMDC");
28     title.addStyleName("label-title");
29     Label mLabel=new Label("m=");
```

```
30      Label nLabel=new Label("n=");
31      final Label cmmdcLabel=new Label();
32      final TextBox mTextBox=new TextBox();
33      final TextBox nTextBox=new TextBox();
34      Button button = new Button("Compute");
35      button.addStyleName("pc-template-btn");
36      VerticalPanel cmmdcPanel = new VerticalPanel();
37      cmmdcPanel.setWidth("100%");
38      cmmdcPanel.setHorizontalAlignment(VerticalPanel.ALIGN_CENTER);
39      cmmdcPanel.add(title);
40      cmmdcPanel.add(mLabel);
41      cmmdcPanel.add(mTextBox);
42      cmmdcPanel.add(nLabel);
43      cmmdcPanel.add(nTextBox);
44      button.addClickHandler(new ClickHandler(){
45        public void onClick(ClickEvent event){
46          String sm=mTextBox.getText();
47          String sn=nTextBox.getText();
48          long m=0,n=0;
49          if(sm.equals("")){
50            Window.alert("\'m\' nu este dat");
51            cmmdcLabel.setText("?");
52            return;
53          }
54          if(sn.equals("")){
55            Window.alert("\'n\' nu este dat");
56            cmmdcLabel.setText("?");
57            return;
58          }
59          try{
60            m=Long.parseLong(sm);
61          }
62          catch(NumberFormatException e){
63            Window.alert("\'m\' nu este numar");
64            cmmdcLabel.setText("?");
65            return;
66          }
67          try{
68            n=Long.parseLong(sn);
69          }
70          catch(NumberFormatException e){
71            Window.alert("\'n\' nu este numar");
72            cmmdcLabel.setText("?");
73            return;
74          }
75          long c=0;
76          if((m!=0)&&(n!=0))
77              getResult(m,n);
78        }
79      });
80      cmmdcPanel.add(button);
81      cmmdcPanel.add(cmmdcLabel);
82      RootPanel.get("cmmdcPage").add(cmmdcPanel);
83    }
84  }
```

Serverul, adică implementarea serviciului este

```
1  package unitbv.cs.td.server;
```

```
2 import com.google.gwt.user.server.rpc.RemoteServiceServlet;
3 import unitbv.cs.td.client.CmmdcService;

5 public class CmmdcServiceImpl extends RemoteServiceServlet
6     implements CmmdcService{
7   public long cmmdc(long m,long n){. . .}
8 }
```

web.xml se completează cu

```
<servlet>
  <servlet-name>cmmdcrpc</servlet-name>
  <servlet-class>unitbv.cs.td.server.CmmdcServiceImpl</servlet-class>
</servlet>

<servlet-mapping>
  <servlet-name>cmmdcrpc</servlet-name>
  <url-pattern>/cmmdcclient/cmmdcrpc</url-pattern>
</servlet-mapping>
```

Test GWT

Generând modulul GWT cu opţiunea suplimentară *-junit*

`webAppCreator -out` *catapp* `-junit` *cale_către\junit*.jar context.MyApp*

în catalogul *catapp* se crează suplimentar structura

```
catapp
   . . .
   |--> test
   |    |--> context
   |    |    |--> client
   |    |    |    |   MyAppTest.java
   |    |    |    MyAppJUnit.gwt.xml
   . . .
```

Obiectivele `ant: test.dev` şi `test.prod` asigură efectuarea testelor prin intermediul lui *junit*.

În cazul unei aplicaţii cu apel de procedură la distanţă GWT sarcina programatorului constă din (cu adaptarea corespunzătoare a denumirii clasei *MyApp*):

- Editarea clasei *MyAppTest* pe baza şablonului prezent;
- Modificarea / adaptarea corespunzătoare a fişierului de configurare *MyAppJUnit.gwt.xml*.

Pentru aplicaţia anterioară codul clasei *unitbv.cs.td.client.CmmdcClientJUnit* este

```
1  package unitbv.cs.td.client;

3  import unitbv.cs.td.shared.FieldVerifier;
4  import com.google.gwt.core.client.GWT;
5  import com.google.gwt.junit.client.GWTTestCase;
6  import com.google.gwt.user.client.rpc.AsyncCallback;
7  import com.google.gwt.user.client.rpc.ServiceDefTarget;

9  public class CmmdcClientTest extends GWTTestCase {

11   public String getModuleName() {
12     return "unitbv.cs.td.CmmdcClientJUnit";
13   }

15   public void testCmmdcService() {
16     // Crearea unui obiect de apelare.
17     CmmdcServiceAsync cmmdcService = GWT.create(CmmdcService.class);
18     ServiceDefTarget target = (ServiceDefTarget) cmmdcService;
19     target.setServiceEntryPoint(GWT.getModuleBaseURL() +
20       "cmmdcclient/cmmdcrpc");
21     delayTestFinish(10000);

23     // Apel server.
24     cmmdcService.cmmdc(56,48, new AsyncCallback<Long>() {
25       public void onFailure(Throwable caught) {
26         fail("Request failure: " + caught.getMessage());
27       }

29       public void onSuccess(Long result) {
30         assertTrue(result.longValue()==8);
31         finishTest();
32       }
33     });
34   }
35 }
```

Fişierul de configurare *CmmdcClientJUnit.gwt.xml* are codul

```
1  <?xml version="1.0" encoding="UTF-8"?>
2  <module>
3    <!-- Inherit our applications main module.        -->
4    <inherits name='unitbv.cs.td.CmmdcClient'/>

6    <!-- Specify the path to any remote services.     -->
7    <servlet path="/cmmdcclient/cmmdcrpc"
8        class="unitbv.cs.td.server.CmmdcServiceImpl" />

10 </module>
```

Exemplul 2.2.4 *Aplicaţie de consultare a unei baze de date. Se consideră o bază de date AgendaEMail alcătuită dintr-un singur tabel adrese (nume varchar(20), email varchar(30)). Se cere realizarea unei aplicaţii de consultare a agendei de adrese e-mail.*

Codul interfeţei

```
1 package unitbv.cs.td.client;
2 import java.util.List;
3 import com.google.gwt.user.client.rpc.RemoteService;

5 public interface AdreseService extends RemoteService{
6     public List<String> getEMail(String nume);
7     public String getNume(String email);
8 }
```

Codul interfeţei asincrone

```
1 package unitbv.cs.td.client;
2 import com.google.gwt.user.client.rpc.AsyncCallback;
3 import java.util.List;

5 public interface AdreseServiceAsync{
6     public void getEMail(String nume,AsyncCallback<List<String>> callback);
7     public void getNume(String email,AsyncCallback<String> callback);
8 }
```

Implementarea interfeţei asincrone este (*AdreseCallback.java*)

```
1 package unitbv.cs.td.client;

3 import com.google.gwt.user.client.rpc.AsyncCallback;
4 import com.google.gwt.core.client.GWT;
5 import com.google.gwt.user.client.Window;
6 import com.google.gwt.user.client.ui.*;
7 import java.util.*;

9 public class AdreseCallback implements AsyncCallback {
10   public void onFailure(Throwable caught) {
11     GWT.log("Error ", caught);
12     caught.printStackTrace();
13     Window.alert(caught.toString());
14   }

16   public void onSuccess(Object result) {
17     final Label labelRez=new Label();
18     Grid grid=new Grid(1,2);
19     if(result instanceof String){
20       String nume=(String)result;
21       final Label labelNume=new Label((String)result);
22       labelRez.setText("Numele cautat : ");
23       grid.setWidget(0,0,labelRez);
24       grid.setWidget(0,1,labelNume);
25       RootPanel.get("rez").add(grid);
26     }
27     if(result instanceof List){
28       final ListBox listAdrese=new ListBox();
29       List<String> list=(List<String>)result;
30       for(int i=0;i<list.size();i++){
31         listAdrese.addItem((String)list.get(i));
32       }
33       labelRez.setText("Adresele cautate : ");
34       grid.setWidget(0,0,labelRez);
35       grid.setWidget(0,1,listAdrese);
36       RootPanel.get("rez").add(grid);
37     }
```

```
38    }
39  }
```

Codul programului client este

```
1  package unitbv.cs.td.client;

3  import com.google.gwt.core.client.EntryPoint;
4  import com.google.gwt.core.client.GWT;
5  import com.google.gwt.user.client.ui.*;
6  import com.google.gwt.user.client.rpc.ServiceDefTarget;
7  import com.google.gwt.event.dom.client.ClickEvent;
8  import com.google.gwt.event.dom.client.ClickHandler;
9  import java.util.*;

11 public class AgendaEMail implements EntryPoint {
12   public void findByNume(String nume){
13     AdreseServiceAsync adreseService=
14       (AdreseServiceAsync)GWT.create(AdreseService.class);
15     ServiceDefTarget target=(ServiceDefTarget)adreseService;
16     String relativeURL=GWT.getModuleBaseURL()+"adrese";
17     target.setServiceEntryPoint(relativeURL);
18     adreseService.getEMail(nume, new AdreseCallback());
19   }

21   public void findByEMail(String email){
22     AdreseServiceAsync adreseService=
23       (AdreseServiceAsync)GWT.create(AdreseService.class);
24     ServiceDefTarget target=(ServiceDefTarget)adreseService;
25     String relativeURL=GWT.getModuleBaseURL()+"adrese";
26     target.setServiceEntryPoint(relativeURL);
27     adreseService.getNume(email, new AdreseCallback());
28   }

30   public void onModuleLoad(){
31     final Button button=new Button("Cauta");
32     final Label labelCriteriu=new Label("Criteriul de cautare");
33     final Label labelEntitate=new Label("Entitatea cautata");
34     final ListBox listBoxCriteriu=new ListBox();
35     final TextBox textBoxEntitate=new TextBox();

37     VerticalPanel adresePanel=new VerticalPanel();
38     Label title=new Label("Agenda de adrese E-MAIL");
39     title.addStyleName("label-title");
40     Grid grid=new Grid(2,2);
41     grid.setWidget(0,0,labelCriteriu);
42     listBoxCriteriu.addItem("nume");
43     listBoxCriteriu.addItem("email");
44     listBoxCriteriu.setVisibleItemCount(1);
45     grid.setWidget(0,1,listBoxCriteriu);
46     grid.setWidget(1,0,labelEntitate);
47     grid.setWidget(1,1,textBoxEntitate);
48     button.addClickHandler(new ClickHandler(){
49       public void onClick(ClickEvent event){
50         String s=textBoxEntitate.getText();
51         if(!"".equals(s)){
52           if(listBoxCriteriu.getSelectedIndex()==0)
53             findByNume(s);
54           else
```

```
55                findByEMail(s);
56            }
57          }
58      });
59      adresePanel.add(title);
60      adresePanel.add(grid);
61      adresePanel.add(button);
62      RootPanel.get("adresePanel").add(adresePanel);
63    }
64  }
```

Partea specifică aplicaţiei din pagina Web (*AgendaTelefonica.html*) este

```
<h1>AgendaEMail</h1>
<p>
<div id="adresePanel"> </div>
</p>
<p>
<div id="rez"> </div>
</p>
```

cu fixarea containerelor în care GWT include *widget*-urile grafice.

Implementarea interfeţei, adică serverul, are codul

```
1  package unitbv.cs.td.server;
2  import unitbv.cs.td.client.*;
3  import java.util.*;
4  import java.sql.*;
5  import com.google.gwt.user.server.rpc.RemoteServiceServlet;

7  public class AdreseServiceImpl extends RemoteServiceServlet
8      implements AdreseService{
9    public static final String DRIVER=
10     "org.apache.derby.jdbc.ClientDriver";
11   public static final String PROTOCOL=
12     "jdbc:derby://localhost:1527/AgendaEMail";

14   public List<String> getEMail(String nume){
15     List<String> adrese;
16     try {
17       Class.forName(DRIVER).newInstance();
18       Connection con = DriverManager.getConnection(PROTOCOL);
19       Statement s = con.createStatement();
20       ResultSet rs =
21         s.executeQuery("SELECT email FROM adrese where nume=\'"+nume+"\'");
22       adrese=new ArrayList<String>();
23       while(rs.next()){
24         adrese.add(rs.getString("email"));
25       }
26     }
27     catch (Exception e) {
28       e.printStackTrace();
29       adrese=null;
30     }
31     finally {
32       try {
33         DriverManager.getConnection("jdbc:derby:;shutdown=true");
34       }
35       catch (SQLException ignore) {}
```

```
36      }
37      return adrese;
38    }

40    public String getNume(String email) {
41      String nume="";
42      try {
43        Connection con = DriverManager.getConnection(PROTOCOL);
44        Statement s = con.createStatement();
45        ResultSet rs =
46          s.executeQuery("SELECT nume FROM adrese where email=\'"+email+"\'");
47        if(rs.next())
48          nume=rs.getString("nume");
49      }
50      catch (Exception e) {
51        e.printStackTrace();
52        nume=null;
53      }
54      finally {
55        try {
56          DriverManager.getConnection("jdbc:derby:;shutdown=true");
57        }
58        catch (SQLException ignore) {}
59      }
60      return nume;
61    }
62  }
```

web.xml se completează cu

```
<servlet>
  <servlet-name>adrese</servlet-name>
  <servlet-class>unitbv.cs.td.server.AdreseServiceImpl</servlet-class>
</servlet>

<servlet-mapping>
  <servlet-name>adrese</servlet-name>
  <url-pattern>/agendaemail/adrese</url-pattern>
</servlet-mapping>
```

În plus, driverul bazei de date trebuie copiat în catalogul `war\WEB-INF\lib`.

2.2.4 Crearea unui widget client

În paralel cu oferta de clase *widget* puse la dispoziţie de GWT există posibilitatea de a crea *widget*-e propri sau client.

O variantă de creare este prezentată în continuare.

O arhitectură posibilă a unei asemenea aplicaţii este dată în Fig. 2.2.

Desfăşurarea bibliotecii de *widget*-e este delimitată prin linii orizontale. *Widget*-ele aparţin pachetului *mywidgets. . . client. MyWidget.css* conţine clasele `css` utilizate de *widget*-uri. Acest fapt este specificat în fişierul de configurare *Widgets.gwt.xml*, având codul

```
1 <?xml version="1.0" encoding="UTF-8"?>
```

```
catapp
   |--> src
----------------------------------------
   |    |--> mywidgets
   |    |    |. . .
   |    |    |    |--> client
   |    |    |    |    |   WidgetClass1.java
   |    |    |    |    |   . . .
   |    |    |    |--> public
   |    |    |    |    |   MyWidget.css
   |    |    |    |    Widgets.gwt.xml
----------------------------------------
   |    |--> context
   |    |    |--> client
   |    |    |    |   MyApp.java
   |    |    |. . .
   |    |--> war
   |    |    |. . .
   |    .classpath
   |    .project
   |    MyApp.launch
   |    README.txt
   |    build.xml
```

Figure 2.2: Aplicaţie GWT cu bibliotecă de *widget*-e.

```
<!DOCTYPE module PUBLIC "-//Google Inc.//DTD Google Web Toolkit 1.6.4//EN"
 "http://google-web-toolkit.googlecode.com/svn/tags/1.6.4/
  distro-source/core/src/gwt-module.dtd">
<module rename-to='mywidgets'>
  <!-- Inherit the core Web Toolkit stuff.                    -->
  <inherits name='com.google.gwt.user.User'/>

  <stylesheet src='MyWidget.css' />
</module>
```

Utilizarea bibliotecii *mywidgets* se declară în fişierul de configurare al aplicaţiei *MyApp.gwt.xml* prin

```
<inherits name='mywidgets...Widgets'/>
```

Caracterul ... corespunde lanţului din definiţia pachetului, mai puţin `client.`

Clasa unui *widget* client extinde clasa `com.google.gwt.user.client.ui.Composite.` Clasa `Composite` trebuie privită ca o clasa acoperitoare unui obiect de tip `Widget.` Elementele care formează interfaţa grafică a *widget*-ului client se includ într-un container reprezentabil grafic (de exemplu `VerticalPanel`). Acest container este încărcat de aplicaţia GWT care utilizează un *widget* client. În codul *widget*-ului client acest lucru este exprimat prin `protected void initWidget(Widget` *container*`).` Această metodă poate fi apelată o singură dată.

Şablonul codului unui *widget* client este

```
package *.client;
import com.google.gwt.user.client.ui.Composite;
. . .
public class ClasaWidget extends Composite {
  public ClasaWidget() {
    . . .
    initWidget(container);
  }
  . . .
}
```

Exemplul 2.2.5 *Se crează o biblioteca alcătuită din două clase widget:*

- *HelloNameWidget - generează un cadru cu un câmp pentru nume şi un buton. După clic pe buton apare mesajul "Hello nume".*
- *CmmdcWidget – aplicaţia anterioară transformată în widget.*

Cele două *widget*-e au codurile, respectiv

```
1  package mywidgets.client;
2  import com.google.gwt.user.client.ui.*;
3  import com.google.gwt.event.dom.client.*;

5  public class HelloNameWidget extends Composite{
6    public HelloNameWidget() {
7      Label title=new Label("Hello Name Widget");
8      title.addStyleName("label-title");
9      Button button = new Button("Apasa-ma");
10     Label nameLabel=new Label("Introduceti numele");
11     final Label sLabel=new Label();
12     final TextBox  nameTextBox=new TextBox();

14     button.addStyleName("button");
15     VerticalPanel vPanel = new VerticalPanel();
16     vPanel.addStyleName("vpanel");
17     vPanel.add(title);
18     vPanel.setWidth("100%");
19     vPanel.setHorizontalAlignment(VerticalPanel.ALIGN_CENTER);
20     vPanel.add(nameLabel);
21     vPanel.add(nameTextBox);
22     vPanel.add(button);
23     vPanel.add(sLabel);
24     button.addClickHandler(new ClickHandler(){
25       public void onClick(ClickEvent event){
26         String name=nameTextBox.getText();
27         sLabel.setText("Hello "+name+" !");
28       }
29     });
30     initWidget(vPanel);
31   }
32 }
```

Şi

```
1  package mywidgets.client;
2  import com.google.gwt.user.client.ui.*;
3  import com.google.gwt.user.client.Window;
4  import com.google.gwt.event.dom.client.*;

6  public class CmmdcWidget extends Composite {
7    public CmmdcWidget() {
8      Label title=new Label("Cmmdc Widget");
9      title.addStyleName("label-title");
10     Label mLabel=new Label("m=");
11     Label nLabel=new Label("n=");
12     Label cmmdcLabel=new Label();
13     TextBox mTextBox=new TextBox();
14     TextBox nTextBox=new TextBox();
15     Button button = new Button("Calculeaza");
16     button.addStyleName("button");

18     VerticalPanel cmmdcPanel = new VerticalPanel();
19     cmmdcPanel.setWidth("100%");
20     cmmdcPanel.addStyleName("vpanel");
21     cmmdcPanel.setHorizontalAlignment(VerticalPanel.ALIGN_CENTER);
22     cmmdcPanel.add(title);
23     cmmdcPanel.add(mLabel);
24     cmmdcPanel.add(mTextBox);
```

```
25      cmmdcPanel.add(nLabel);
26      cmmdcPanel.add(nTextBox);
27      CmmdcWidgetClickHandler clickHandler=
28        new CmmdcWidgetClickHandler(mTextBox,nTextBox,cmmdcLabel);
29      button.addClickHandler(clickHandler);
30      cmmdcPanel.add(button);
31      cmmdcPanel.add(cmmdcLabel);

33      initWidget(cmmdcPanel);
34    }
35  }

37  class CmmdcWidgetClickHandler implements ClickHandler{. . .}
```

Codul aplicaţiei GWT care foloseşte *widget*-ele client definite anterior este

```
1  package unitbv.cs.td.client;
2  import com.google.gwt.core.client.EntryPoint;
3  import com.google.gwt.user.client.ui.*;
4  import mywidgets.client.*;

6  public class MyApp implements EntryPoint {

8    public void onModuleLoad() {
9      HorizontalPanel hPanel=new HorizontalPanel();
10     hPanel.setSpacing(50);
11     CmmdcWidget cw=new CmmdcWidget();
12     HelloNameWidget hnw=new HelloNameWidget();
13     hPanel.add(hnw);
14     hPanel.add(cw);
15     RootPanel.get("ginterface").add(hPanel);
16   }
17 }
```

2.2.5 Încărcarea unui fişier - GWT Upload

GWT oferă o soluţie prefabricată problemei încărcării unui fişier al clientului pe calculatorul server.

Aplicaţia GWT corespunde clientului care alege fişierul de încărcat şi transmite datele unui server receptor, reprezentat, în cazul exemplului dezvoltat, de un servlet. Clasele *widget* ajutătoare sunt `FormPanel` şi `FileUpload`. Într-un container `FormPanel` pot fi incluse *widget*e de tip `TextBox, PasswordTextBox, RadioButton, CheckBox, TextArea, FileUpload, Hidden`.

Dintre metodele care intervin la încărcarea unui fişier amintim:

- `public void setAction(String` *url*`)`
- `public void setMethod(FormPanel.METHOD_POST)`

  În alt context se poate folosi `FormPanel.METHOD_GET`.

- `public void setEncoding(FormPanel.ENCODING_MULTIPART)`

  În alt context se poate folosi `FormPanel.ENCODING_URLENCODED.`

- `public void submit()`

- `HandlerRegistration addSubmitHandler(FormPanel.SubmitHandler` *handler*`)`

  Interfaţa `SubmitHandler`

- `HandlerRegistration addSubmitCompleteHandler(FormPanel.SubmitCompleteHandler` *handler*`)`

Fiecare din interfeţele `SubmitHandler` şi `SubmitCompleteHandler` declară metoda `void onSubmit(FormPanel.SubmitEvent` *event*`)` în care se programează activităţile înaintea expedierii formularului şi respectiv, după recepţia răspunsului furnizat de servlet.

*Widget*ul `UploadFile` afişează o fereastră de dialog prin care se selectează fişierul ce umează a fi încărcat.

Codul clasei client este

```
1  package unitbv.cs.td.client;

3  import com.google.gwt.core.client.*;
4  import com.google.gwt.user.client.Window;
5  import com.google.gwt.user.client.ui.*;
6  import com.google.gwt.event.dom.client.ClickEvent;
7  import com.google.gwt.event.dom.client.ClickHandler;

9  public class Upload implements EntryPoint{
10   public void onModuleLoad(){
11     final FormPanel form = new FormPanel();
12     form.setAction("http://localhost:8080/upload/upload");
13     form.setEncoding(FormPanel.ENCODING_MULTIPART);
14     form.setMethod(FormPanel.METHOD_POST);

16     Label title=new Label("File upload");
17     title.addStyleName("label-title");

19     VerticalPanel panel = new VerticalPanel();
20     form.setWidget(panel);

22     panel.add(title);

24     // Crearea unui widget FileUpload
25     FileUpload upload = new FileUpload();
26     upload.setName("uploadFormElement");
27     panel.add(upload);

29     // Adaugarea unui buton "submit"
30     panel.add(new Button("Submit", new ClickHandler() {
31       public void onClick(ClickEvent event) {
```

```
32          form.submit();
33        }
34      }));

36      // Activitati premergatoare expedierii formularului.
37      form.addSubmitHandler(new FormPanel.SubmitHandler() {
38        public void onSubmit(FormPanel.SubmitEvent event) {
39        }
40      });

42      // Activitati la receptionarea raspunsului
43      form.addSubmitCompleteHandler(new FormPanel.SubmitCompleteHandler() {
44        public void onSubmitComplete(FormPanel.SubmitCompleteEvent event) {
45          String results=event.getResults();
46          Window.alert(results);
47        }
48      });
49      RootPanel.get().add(form);
50    }
51  }
```

Servlet-ul utilizează pachetele *apache commons-fileupload, commons-io* şi se va instala în catalogul *upload* a serverului Web *apache-tomcat.* Fişierele încărcate de client se vor salva în catalogul *. . . \webapps\upload\upload.* Codul servet-ului este

```
1  package upload;
2  import java.io.IOException;
3  import java.io.InputStream;
4  import java.io.FileOutputStream;
5  import java.io.File;
6  import javax.servlet.ServletException;
7  import javax.servlet.http.HttpServlet;
8  import javax.servlet.http.HttpServletRequest;
9  import javax.servlet.http.HttpServletResponse;
10 import javax.servlet.ServletOutputStream;
11 import javax.servlet.annotation.WebServlet;
12 import java.util.List;
13 import java.util.Iterator;
14 import java.util.Vector;
15 import org.apache.commons.fileupload.servlet.ServletFileUpload;
16 import org.apache.commons.fileupload.FileItemIterator;
17 import org.apache.commons.fileupload.FileItemStream;

19 @WebServlet(urlPatterns = "/upload")

21 public class FileUploadServlet extends HttpServlet{

23   public void doPost(HttpServletRequest req,HttpServletResponse res)
24       throws ServletException,IOException {
25     String fs=System.getProperty("file.separator");
26     String pathTomcat = new File(".").getCanonicalPath();
27     String contextPath=req.getContextPath();

29     res.setContentType("text/plain");
30     ServletOutputStream out = res.getOutputStream();
31     try{
32       ServletFileUpload upload = new ServletFileUpload();
33       FileItemIterator iter = upload.getItemIterator(req);
```

```
34        while (iter.hasNext()) {
35          FileItemStream item = iter.next();
36          String name = item.getFieldName();
37          out.println(name);
38          if (!item.isFormField()) {
39            String fileName = item.getName();
40            out.println(fileName);
41            InputStream in=item.openStream();
42            File file=new File(pathTomcat+fs+"webapps"+
43              contextPath+fs+"upload"+fs+fileName);
44            byte[] b=new byte[1024];
45            FileOutputStream fos=new FileOutputStream(file);
46            int s=0;
47            do{
48              s=in.read(b,0,1024);
49              if(s!=-1)
50                fos.write(b,0,s);
51            }
52            while(s!=-1);
53            fos.close();
54            in.close();
55            out.println("The file "+fileName+" has been uploaded !");
56            out.close();
57          }
58        }
59      }
60      catch(Exception e){
61        System.out.println("Exception: "+e.getMessage());
62      }
63    }
64  }
```

Pentru utilizarea aplicaţiei client în modul Web, după compilare, conţinutul catalogului *war* se copiază în catalogul *...\webapps* din *apache-tomcat.* Dintr-un navigator, clientul se apelează prin
`http://host:8080/uploadGwt/UploadGwt.html`.

2.2.6 GWT prin *Google AppEngine*

Integrarea unei aplicaţii client GWT în platforma *Google AppEngine* (GAE) de Cloud Computing, necesită executarea următoarelor operaţii:

1. Se compilează aplicaţia GWT în vederea utilizării în modul Web;

2. Catalogul `war` se completează cu fişierul `appengine-web.xml` plasându-l în catalogul *WEB-INF;*

3. Catalogul `war` se redenumeşte `www`.

Din catalogul care conţine catalogul `www` se lansează simulatorul GAE.

Exemplul 2.2.6 *Integrarea aplicaţiei GWT de calcul a celui mai mare divizor comun în platforma Google de Cloud Computing.*

În urma operaţiilor de mai sus rezultă structura

```
gwtcmmdc
|--> www
|    |--> cmmdc
|    |    | . . .
|    |--> WEB-INF
|    |    |--> classes
|    |    |    | . . .
|    |    |--> lib
|    |    |    | . . .
|    |    |    appengine-web.xml
|    |    |    web.xml
|    |    Cmmdc.html
|    |    Cmmdc.css
```

După lansarea simulatorului GAE, aplicaţia se apelează prin `http://localhost:8080/Cmmdc.html`.

Partea II

SERVICII WEB

Capitolul 3

Servicii JAX-WS

Un serviciu Web este o aplicaţie client-server cu serverul găzduit de un server Web, apelabil prin aplicaţia client. Protocolul de comunicaţie este `http`.

Sunt cunoscute următoarele tipuri de servicii Web:

- Servicii bazate pe modelul Remote Precedure Call (RPC) - Apel de Procedură de la Distanţă.

  Protocolul de reprezentare a cererii şi a răspunsului (de serializare / deserializare) variază. Din acest punct de vedere sunt cunoscute:

  - Servicii *xml-rpc* (www.xmlrpc.org). Cererea şi răspunsul sunt transmise prin cod `xml` cuprins în corpul mesajului `http`.
  - Servicii *json-rpc* (www.json-rpc.org) bazat pe reprezentarea JSON.
  - Servicii *hessian.* Se utilizează un protocol pentru serializare / deserealizare bazat pe reprezentarea binară a datelor. Protocolul a fost dezvoltat de firma *Caucho Technologies* (2007).
  - Servicii bazate pe interfaţa de programare *Java API for XML Web Services (JAX-WS).* Protocolul de reprezentare este *Simple Object Access Protocol,* (SOAP) standard W3C (World Wide Web Consortium). Serviciul este descris printr-un document *Web Service Description Language* (WSDL), de tip `xml`. Interfaţa de programare JAX-WS este varianta cea mai recentă pentru serviciile cunoscute sub numele de servicii *soap-rpc.* Aspectele privind SOAP sunt complet transparente programatorului.

  Pentru fiecare caz semnalat mai sus sunt realizate implementări specifice în mai multe limbaje / platforme de programare.

- Servicii REST.

  REpresentational State Transfer (REST) este un model de arhitectură de aplicaţie distribuită[1].

  REST specifică modul cum o resursă - entitate care conţine informaţie specifică - este definită şi cum poate fi adresată.

  Identificarea unei resurse se face printr-un URI (*Universal Resource Identifier*).

  Transferul resursei către un client sau prelucrarea resursei se face utilizând o interfaţă de programare care conţine o mulţime de operaţii *GET, POST, PUT, DELETE.*

3.1 Descrierea unui serviciu prin WSDL

Realizarea şi gestionarea serviciilor Web de tip JAX-WS necesită familiarizarea cu

- *XML Schema*
- *WSDL* – Web Service Description Language

Spre deosebire de aplicaţiile bazate pe apelul de procedură la distanţă (RMI, CORBA,) unde prezentarea ofertei se face printr-o interfaţa Java, într-un serviciu JAX-WS posibilităţile oferite sunt descrise într-un fişier WSDL.

3.1.1 XML Schema

O schemă este un model pentru descrierea structurii informaţiei. În contextul `xml`, o schemă descrie un model pentru o familie de documente `xml`.

Primul obiectiv al unei scheme este de a permite validarea automată a structurii unui document `xml`.

Familiarizarea cu elementele de conţinut şi sintaza *XML Schema* se va începe cu un exemplu:

Documentul `struct.xml`

```
1 <?xml version="1.0" encoding="UTF-8"?>
2 <mk:structura xmlns:mk="http://www.distr2.edu"
3   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
4   xsi:schemaLocation="http://www.distr2.edu struct-schema.xsd">
```

[1] REST a fost introdus de Roy Fielding, în teza sa de doctorat din 2000. Roy Fielding este autorul principal al specificaţiilor protocolului http.

```
  <mk:disciplina fel="obligatoriu">
    <mk:nume> Analiza numerica </mk:nume>
    <mk:fond-de-timp>
        <mk:curs>2</mk:curs>
        <mk:seminar>1</mk:seminar>
        <mk:laborator>1</mk:laborator>
    </mk:fond-de-timp>
  </mk:disciplina>
  <mk:disciplina fel="obligatoriu">
    <mk:nume> Programare distribuita </mk:nume>
    <mk:fond-de-timp>
        <mk:curs>2</mk:curs>
        <mk:seminar>0</mk:seminar>
        <mk:laborator>2</mk:laborator>
    </mk:fond-de-timp>
  </mk:disciplina>
  <mk:disciplina fel="obligatoriu">
    <mk:nume> Soft matematic </mk:nume>
    <mk:fond-de-timp>
        <mk:curs>2</mk:curs>
        <mk:seminar>0</mk:seminar>
        <mk:laborator>1</mk:laborator>
    </mk:fond-de-timp>
  </mk:disciplina>
</mk:structura>
```

de exemplu, are schema *struct-schema.xsd*

```
<?xml version="1.0" encoding="UTF-8"?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:mk="http://www.distr2.edu"
 targetNamespace="http://www.distr2.edu">

  <xsd:element name="structura">
    <xsd:complexType>
      <xsd:choice minOccurs="0" maxOccurs="unbounded">
        <xsd:element ref="mk:disciplina"/>
      </xsd:choice>
    </xsd:complexType>
  </xsd:element>

  <xsd:element name="disciplina">
    <xsd:complexType mixed="true">
      <xsd:sequence>
        <xsd:element ref="mk:nume"/>
        <xsd:element ref="mk:fond-de-timp"/>
      </xsd:sequence>
      <xsd:attribute name="fel" type="xsd:string" use="required"/>
    </xsd:complexType>
  </xsd:element>

  <xsd:element name="nume" type="xsd:string"/>

  <xsd:element name="fond-de-timp">
    <xsd:complexType mixed="true">
      <xsd:sequence>
        <xsd:element ref="mk:curs"/>
        <xsd:element ref="mk:seminar"/>
```

```
31          <xsd:element ref="mk:laborator"/>
32        </xsd:sequence>
33      </xsd:complexType>
34    </xsd:element>

36    <xsd:element name="curs" type="xsd:string"/>

38    <xsd:element name="seminar" type="xsd:string"/>

40    <xsd:element name="laborator" type="xsd:string"/>
41  </xsd:schema>
```

Alte marcaje utilizabile sunt:

- ```
  <xsd:simpleType name=". . ." >
    <xsd:restriction base="xsd:string" >
      <xsd:maxLength value="n" />
      <xsd:pattern value="[0-9]" />
    </xsd:restriction>
  </xsd:simpleType>
  ```

- ```
  <xsd:complexType name="aaa" >
    <xsd:sequence>
      <xsd:element name="bbb" type="ccc"
        minOccurs="0"  maxOccurs="unbounded" />
    </xsd:sequence>
  </xsd:complexType>
  ```

Tipuri simple: string, byte, integer, decimal, boolean, time, date, etc.
Grupuri. Se pot defini grupuri de elemente şi de atribute

```
<xsd:group name="tipElementDisciplina" >
   <xsd:sequence>
      <xsd:element name="nume" type="tipNume" />
      <xsd:element name="fond-de-timp" type="tipFondTimp" />
   </xsd:sequence>
</xsd:group>

<xsd:attributeGroup name="atribDisciplina" >
   <xsd:attribute name="fel" type="xsd:string" use="required" />
   <xsd:attribute name="finalizare" type="xsd:string" />
</xsd:attributeGroup>

<xsd:complexType name="tipdisciplina" >
   <xsd:sequence>
      <xsd:group ref="tipElementDisciplina" />
      <xsd:attributeGroup ref="atribDisciplina" />
   <xsd:sequence>
</xsd:complexType>
```

Comentariile

- destinate a fi citite de oameni se introduc în marcajul `xsd:documentation;`
- destinate a fi procesate se introduc în marcajul `xsd:appinfo`

Ambele marcaje trebuie inglobate în `xsd:annotation.`

```
<xsd:annotation>
  <xsd:documentation xml:lang="en">
     text
  </xsd:documentation>
  <xsd:appinfo source=". . ." >
     <bind xmlns=". . ." >
        <class name=". . ." />
     </bind>
  </xsd:appinfo>
</xsd:annotation>
```

Schemele se pot compune prin

```
<xsd:include schemaLocation="fisier.xsd" />
```

Pentru a evita conflictul între numele atribuite diverselor elemente se introduc:

- Spaţiul de nume - namespace - definit ca un şir de caractere sub forma unui URI (Universal Resource Identifier);
- Numele calificat - *QName* qualified name - alcătuit dintr-un nume local asociat cu namespace.

Procesarea documentelor `xml` face apel la *numele calificat* al unei entităţi.
Un spaţiu de nume se defineşte prin

```
<xsd:schema targetNamespace="numeleSpatiuluiDeNume"
   xmlns:prefix="numeleSpatiuluiDeNume"
   xmlns:xsd="http://www.w3.org/2001/XMLSchema">
```

Prin folosirea prefixului se specifică spaţiul de nume al fiecărei entităţi utilizate.

3.1.2 WSDL

WSDL (*Web Service Description Language*) este un limbaj `xml` pentru descrierea serviciilor Web.
Presupunem existenţa următoarelor entităţi

```
|| Server Web la adresa http://localhost:8080
||      Serviciu Web /CmmdcWS
||          Operatie
||             Nume: cmmdc
||          Operatie
||             Nume: . . .
```

Serviciul se va apela prin *http://localhost:8080/CmmdcWS.* Această referinţă se numeşte *punct final - endpoint.*

Pentru a evita conflictul între numele atribuite diverselor elemente se utilizează denumiri calificate. iar structura de mai sus devine

```
|| Server Web la adresa http://localhost:8080
||      Serviciu Web /CmmdcWS
||          Operatie
||             Nume local: cmmdc
||             Namespace:  http://cs.unitbv.ro/ws
||          Operatie
||             Nume local: . . .
```

Pentru acest exemplu, operaţia este definită de o metodă având doi parametrii şi returnează un rezultat:

```
|| Operatie
||    Nume local: cmmdc
||    Namespace:  http://cs.unitbv.ro/ws
||    Parametrii:
||         m: string
||         n: string
||    Returneaza:
||         string
```

Tipurile (string, int, long, etc) utilizate sunt precizate în spaţiul de nume `http://www.w3.org/2001/XMLSchema.`
Definirea operaţiei devine

```
|| Operatie
||    Nume local: cmmdc
||    Namespace:  http://cs.unitbv.ro/ws
||    Parametrii:
||         m: string din http://www.w3.org/2001/XMLSchema
||         n: string din http://www.w3.org/2001/XMLSchema
||    Returneaza:
||         string din http://www.w3.org/2001/XMLSchema
```

În terminologia serviciilor Web RPC se utilizează

- mesaj de intrare - input message - pentru datele de apelare a unei metode;
- parte - part - pentru un parametru al mesajului de intrare;
- mesaj de ieşire - output message - pentru datele returnate de operaţie.

Asfel vom avea

```
|| Operatie
||    Nume local: cmmdc
||    Namespace:  http://cs.unitbv.ro/ws
||    Input message:
||       Part 1:
||          Name: m
||          Type: string din http://www.w3.org/2001/XMLSchema
```

```
||       Part 2:
||          Name: n
||          Type: string din http://www.w3.org/2001/XMLSchema
||    Output message:
||       Part:
||          Name: return
||          Type: string din http://www.w3.org/2001/XMLSchema
```

Codul unui mesaj de intrare poate fi

```
<pre:cmmdc xmlns:pre="http://cs.unitbv.ro/ws">
   <m>56</m>
   <n>48</n>
</pre:cmmdc>
```

iar mesajul de iesire poate fi

```
<pre:cmmdc xmlns:pre="http://cs.unitbv.ro/ws">
   <return>8</return>
</pre:cmmdc>
```

XMLSchema permite definirea de tipuri complexe:

```
<xsd:schema targetNamespace="http://cs.unitbv.ro/ws"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema">
  <xsd:element name="cmmdcRequest">
     <xsd:complexType>
        <xsd:element name="m" type="xsd:string"/>
        <xsd:element name="n" type="xsd:string"/>
     </xsd:complexType>
  </xsd:element>
</xsd:schema>
```

Definirea operaţiei va constă din

1. schema

   ```
   <xsd:schema targetNamespace="http://cs.unitbv.ro/ws"
     xmlns:xsd="http://www.w3.org/2001/XMLSchema">
     <xsd:element name="cmmdcRequest">
        <xsd:complexType>
           <xsd:element name="m" type="xsd:string"/>
           <xsd:element name="n" type="xsd:string"/>
        </xsd:complexType>
     </xsd:element>
   </xsd:schema>
   ```

2. operaţia propriu-zisă

   ```
   || Operatie
   ||    Nume local: cmmdc
   ||    Namespace:  http://cs.unitbv.ro/ws
   ||    Input message:
   ||       Part 1:
   ||          Name: cerereCmmdc
   ||          Type: cmmdcRequest din http://cs.unitbv.ro/ws
   ||    Output message:
   ||       Part 1:
   ||          Name: return
   ||          Type: string din http://www.w3.org/2001/XMLSchema
   ```

iar mesajul de intrare va fi

```
<pre:cmmdcRequest xmlns:pre="http://cs.unitbv.ro/ws">
   <m>56</m>
   <n>48</n>
</pre:cmmdcRequest>
```

Extindem incluzând şi răspunsul oferit de operaţie

1. schema

```
<xsd:schema targetNamespace="http://cs.unitbv.ro/ws"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema">
  <xsd:element name="cmmdcRequest">
     <xsd:complexType>
        <xsd:element name="m" type="xsd:string"/>
        <xsd:element name="n" type="xsd:string"/>
     </xsd:complexType>
  </xsd:element>
  <xsd:element name="responseCmmdc" type="xsd:string"/>
</xsd:schema>
```

2. operaţia propriu-zisă

```
|| Operatie
||    Nume local: cmmdc
||    Namespace:  http://cs.unitbv.ro/ws
||    Input message:
||       Part 1:
||          Name: cerereCmmdc
||          Type: cmmdcRequest din http://cs.unitbv.ro/ws
||    Output message:
||       Part 1:
||          Name: raspunsCmmdc
||          Type: responseCmmdc din http://cs.unitbv.ro/ws
```

Răspunsul va fi

```
<pre:cmmdcResponse xmlns:pre="http://cs.unitbv.ro/ws">
   8
</pre:cmmdcResponse>
```

Acest mod de definire a unui serviciu Web poartă numele de *stilul document*[2]. Acest stil este impus de WS-I (Web Services Interoperability organization).

Determinarea operaţiei apelate se face exclusiv pe baza tipului din mesajul de intrare. Nu pot exista două operaţii cu datele de intrare definite prin acelaşi nume calificat.

Asamblând, se obţine

[2] Un alt mod de definire, nerecomandat în prezent, este *stilul RPC*.

```
|| Server Web la adresa http://localhost:8080
||    Schema
||         . . .
||    Operatie
||       Nume local: cmmdc
||       Namespace:  http://cs.unitbv.ro/ws
||       Input message:
||          Part 1:
||             Name: cerereCmmdc
||             Type: cmmdcRequest din http://cs.unitbv.ro/ws
||       Output message:
||          Part 1:
||             Name: raspunsCmmdc
||             Type: responseCmmdc din http://cs.unitbv.ro/ws
```

PortType. Operaţiile sunt grupate în colecţii numite *PortType.* Un PortType este denumită printr-un nume calificat (nume local şi namespace).

```
|| Server Web la adresa http://localhost:8080
||    Schema
||         . . .
||
||    PortType
||       Nume local: functiiWS
||       Namespace:  http://cs.unitbv.ro/ws
||
||       Operatie
||         Nume local: cmmdc
||         Namespace:  http://cs.unitbv.ro/ws
||         Input message:
||           Part 1:
||             Name: cerereCmmdc
||             Type: cmmdcRequest din http://cs.unitbv.ro/ws
||         Output message:
||           Part 1:
||             Name: raspunsCmmdc
||             Type: responseCmmdc din http://cs.unitbv.ro/ws
||
||       Operatie
||         . . .
||
||    PortType
||      . . .
```

Binding. Prin *binding* se specifică protocolul utilizat pentru prelucrarea şi transmiterea mesajelor. Cel mai utilizat protocol pentru prelucrarea mesajelor - aproape complet transparent programatorului - este *Simple Object Access Protocol* SOAP, iar pentru transportul mesajelor este *Hyper Text Transport Protocol* HTTP.

```
|| Name      :  binding1
|| Port type :  functiiWS
|| Format    :  SOAP
|| Transport :  HTTP
```

Port. Serviciul se poate instala / desfăşura pe mai multe calculatoare. Fiecare asemenea calculator devine un port. Implementarea propriu-zisă a operaţiilor poate fi diferită - chiar în limbaje de programare diferite.

Astfel, obţinem următoarea descriere a serviciului

```
|| Serviciu Web
||    Schema
||         . . .
||
||    PortType
||       Nume local: functiiWS
||       Namespace:  http://cs.unitbv.ro/ws
||
||       Operatie
||         . . .
||
||    PortType
||      . . .
||
||    Binding
||      Name      :  binding1
||      Port type :  functiiWS
||      Format    :  SOAP
||      Transport :  HTTP
||    . . .
||
||    Port
||      Name    :  port1
||      Binding :  binding1
||      Endpoint:
||    . . .
```

Toate noţiunile introduse pentru descrierea serviciului Web vor face parte dintr-un acelaşi spaţiu de nume *targetNamespace*

```
|| Serviciu Web
|| Target namespace: http://cs.unitbv.ro/ws
||    Schema
||         . . .
||
||    PortType
||       Nume local: functiiWS
||       Namespace:  http://cs.unitbv.ro/ws
||
||       Operatie
||         . . .
||
||    PortType
||      . . .
||
||    Binding
||      Name      :  binding1
||      Port type :  functiiWS
||      Format    :  SOAP
||      Transpoat :  HTTP
||    . . .
||
||    Port
```

```
||      Name    :  port1
||      Binding :  binding1
||      Endpoint:
||    . . .
```

Aceasta descriere a unui serviciu Web corespunde standardului *WSDL.* Structura unui document `wsdl` cuprinde

```
<definitions>
  <types>
     Definirea tipurilor de date utilizate
  </types>
  <message>
     Definirea mesajelor utilizate. Un mesaj corespunde
     parametrilor sau rezultatelor functiilor ce compun
     serviciul.
  </message>
  <portType>
     Declara functiile serviciului.
     Un port defineste un punct de conexiune cu serviciul Web.
  </portType>
  <binding>
     Declara protocoalele utilizate de serviciul web.
  </binding>
</definitions>
```

Pentru un serviciu Web, documentul `wsdl` generat depinde de produsul informatic care sustine serviciul Web. Funcţie de acest produs, sunt introduse în documentul `wsdl` şi alte elemente.

Exemplul 3.1.1 *Fişierul* `wsdl` *pentru serviciul Web de calcul a celui mai mare divizor comun a două numere naturale generat de Metro.*

```
1  <!--
2    Published by JAX-WS RI at http://jax-ws.dev.java.net.
3      RI's version is Metro/2.3 (tags/2.3-7528; 2013-04-29T19:34:10+0000)
4      JAXWS-RI/2.2.8 JAXWS/2.2 svn-revision#unknown.
5  -->
6  <!--
7    Generated by JAX-WS RI at http://jax-ws.dev.java.net.
8      RI's version is Metro/2.3 (tags/2.3-7528; 2013-04-29T19:34:10+0000)
9      JAXWS-RI/2.2.8 JAXWS/2.2 svn-revision#unknown.
10 -->
11 <definitions
12   xmlns:wsu="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-
```

```
13        wssecurity-utility-1.0.xsd"
14    xmlns:wsp="http://www.w3.org/ns/ws-policy"
15    xmlns:wsp1_2="http://schemas.xmlsoap.org/ws/2004/09/policy"
16    xmlns:wsam="http://www.w3.org/2007/05/addressing/metadata"
17    xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
18    xmlns:tns="http://server.cmmdc/"
19    xmlns:xsd="http://www.w3.org/2001/XMLSchema"
20    xmlns="http://schemas.xmlsoap.org/wsdl/"
21    targetNamespace="http://server.cmmdc/" name="CmmdcWSService">
22    <types>
23      <xsd:schema>
24      <xsd:import namespace="http://server.cmmdc/"
25      schemaLocation=
26        "http://localhost:8080/jaxws-cmmdc/cmmdcws?xsd=1" />
27      </xsd:schema>
28    </types>
29    <message name="cmmdc">
30       <part name="parameters" element="tns:cmmdc" />
31    </message>
32    <message name="cmmdcResponse">
33       <part name="parameters" element="tns:cmmdcResponse" />
34    </message>
35    <portType name="CmmdcWS">
36      <operation name="cmmdc">
37        <input wsam:Action="http://server.cmmdc/CmmdcWS/cmmdcRequest"
38          message="tns:cmmdc" />
39        <output wsam:Action="http://server.cmmdc/CmmdcWS/cmmdcResponse"
40          message="tns:cmmdcResponse" />
41      </operation>
42    </portType>
43    <binding name="CmmdcWSPortBinding" type="tns:CmmdcWS">
44      <soap:binding
45         transport="http://schemas.xmlsoap.org/soap/http"
46         style="document" />
47      <operation name="cmmdc">
48        <soap:operation soapAction="" />
49        <input>
50          <soap:body use="literal" />
51        </input>
52        <output>
53          <soap:body use="literal" />
54        </output>
55      </operation>
56    </binding>
57    <service name="CmmdcWSService">
58      <port name="CmmdcWSPort"
59        binding="tns:CmmdcWSPortBinding">
60        <soap:address
61          location="http://localhost:8080/jaxws-cmmdc/cmmdcws" />
62      </port>
63    </service>
64  </definitions>
```

3.2 Modelul JAX-WS prin Metro

JSR (Java Specification Request) 109 defineşte o interfaţă de programare (API) pentru realizarea serviciilor Web bazate pe RPC : *Java API for XML Web Services (JAX-WS).* Un asemenea serviciu Web se poate implementa prin:

- servlet:

  Serviciul este implementat ca o clasă Java care rulează într-un container Web, fiind integrat într-un servlet. Integrarea este complet transparentă programatorului.

- sesiune EJB (Enterprise Java Bean) fără stare (stateless session):

  Serviciul rulează într-un container EJB.

3.2.1 Serviciu Web ca servlet în *apache-tomcat* prin *ant*

Cadrul de lucru pe care îl vom utiliza este *Metro,* dezvoltat de *Oracle.* Un scop al cadrului de lucru *Metro* este asigurarea interoperabilităţii între server şi client atunci când acestea sunt realizate pe platformele soft Java şi .NET, dar facilitează şi dezvoltarea în mediul omogen Java. *Metro* implementează modelul JAX-WS.

Alternativ, s-ar fi putut folosi implementarea de referinţă *jaxws-ri,* dezvoltat tot de *Oracle* sau *apache-CXF.*

Metro oferă suport pentru dezvoltarea serviciului Web pe serverele:

- *apache-tomcat;*[3]
- *glassfish.*

Instalarea se face

- prin `ant` cu fişierul *metro-on-tomcat.xml* aflat în distribuţia lui *Metro.*
- fişierele `jar` aflate în catalogul *metro\lib* se copiază în *apache-tomcat-7.*.*\lib.*

Dezvoltarea aplicaţiei server

Clasa serverului este o clasă POJO cu adnotări specifice. Vom dezvolta serviciul Web pentru calculul celui mai mare divizor comun a două numere naturale.

[3]Funcţionează şi în serverul Web `jetty`.

Definirea serviciului, a operaţiilor pe care le oferă serviciul şi a parametrilor de intrare pentru fiecare operaţie se face utilizând *adnotările* `@WebService`, respectiv `@WebMethod` şi `@WebParam`.

Structura clasei server va fi

```
1  package cmmdc.server;
2  import javax.jws.WebMethod;
3  import javax.jws.WebParam;
4  import javax.jws.WebService;

6  @WebService()
7  public class CmmdcWS {

9    @WebMethod
10   public long cmmdc(@WebParam(name = "m") long m,
11                     @WebParam(name = "n") long n) { }
12 }
```

Programatorul va completa codul metodei *cmmdc.*

Vom considera structura

```
myapp
|--> src
|      |--> mypackage
|      |      |--> server
|      |      |      |  *.java
|--> war
|      |--> WEB-INF
|      |      |--> classes
|      |      |--> lib
|      |      |      web.xml
|      |      |      sun-jaxws.xml
```

Compilarea se face prin intermediul utilitarului `apt - Annotation Proccessing Tool` din *jdk* prin intermediul unei sarcini date de clasa `com.sun.tools.ws.ant.Apt` din Metro. Codul corespunzător din `build.xml` este

```
<taskdef name="apt" classname="com.sun.tools.ws.ant.Apt">
      <classpath refid="myclasspath"/>
</taskdef>

<target name="build-server" depends="init">
      <apt
        fork="true"
        debug="true"
        destdir="war/WEB-INF/classes"
        sourcedestdir="war/WEB-INF/classes"
        sourcepath="src">
        <classpath>
          <path refid="myclasspath"/>
        </classpath>
        <source dir="src">
          <include name="**/server/*.java"/>
        </source>
      </apt>
 </target>
```

Pe baza claselor din `source`=*src\mypackage\server* se vor genera în `sourcedestdir` o serie de clase iar rezultatul compilării se depun în catalogul indicat de `destdir`.

În cazul exemplului considerat aceste clase sunt *cmmdc.server.jaxws. Cmmdc.java* şi *cmmdc.server.jaxws.CmmdcResponse.java.*

Sunt necesare două fişiere de configurare

- *sun-jaxws.xml*

```
1 ?xml version="1.0" encoding="UTF-8"?>

3 <endpoints xmlns='http://java.sun.com/xml/ns/jax-ws/ri/runtime'
4     version='2.0'>
5   <endpoint
6     name='jaxws-cmmdc'
7     implementation='cmmdc.server.CmmdcWS'
8     url-pattern='/cmmdcws'/>
9 </endpoints>
```

- *web.xml*

```
1 ?xml version="1.0" encoding="UTF-8"?>

3 <web-app version="2.4" xmlns="http://java.sun.com/xml/ns/j2ee">
4   <listener>
5     <listener-class>
6     com.sun.xml.ws.transport.http.servlet.WSServletContextListener
7     </listener-class>
8   </listener>
9   <servlet>
10    <servlet-name>cmmdcws</servlet-name>
11    <servlet-class>
12       com.sun.xml.ws.transport.http.servlet.WSServlet
13    </servlet-class>
14    <load-on-startup>1</load-on-startup>
15  </servlet>
16  <servlet-mapping>
17    <servlet-name>cmmdcws</servlet-name>
18    <url-pattern>/cmmdcws</url-pattern>
19  </servlet-mapping>
20  <session-config>
21    <session-timeout>60</session-timeout>
22  </session-config>
23 </web-app>
```

Se reţin următoarele corelaţii ale denumirilor:

- *url-pattern* fixat în *web.xml* este redeclarat în fişierele *sun-jaxws.xml.*
- Numele serviciului declarat în *sun-jaxws.xml* trebuie să fie numele arhivei `war`.

  Pentru exemplul nostru, *wsdl*-ul serviciului va fi disponibil la

http://host:8080/jaxws-cmmdc/cmmdcws?wsdl

Dezvoltarea aplicaţiei client. Realizarea aplicaţiei client presupune ca serviciul să fie activ pe serverul Web.

Dezvoltarea părţii (aplicaţiei) client începe cu generarea unor clase care mijlocesc apelarea serviciului. Generarea se face utilizând utilitarul `wsimport` din `jdk` pe baza accesării fişierului `wsdl` asociat serviciului.

Codul obiectivului din `build.xml` este

```
<target name="generate-client" depends="init">
     <exec executable="${env.JAVA_HOME}/bin/wsimport">
          <arg value="-d"/>
          <arg value="${build.dir}"/>
          <arg value="-keep"/>
          <arg value="-p"/>
          <arg value="${app.name}.client"/>
          <arg value="${wsdl.uri}"/>
     </exec>
</target>
```

Opţiunea `-d` specifică locaţia unde se depun fişierele generate, opţiunea `-p` indică pachetul din care fac parte clasele generate.

Dintre aceste clase, în codul clientului propriu-zis se foloseşte clasa *CmmdcWSService.* Numele clasei s-a obţinut adăugând sufixul *Service* la numele clasei server. Această clasă conţine metoda *getCmmdcWSPort()* ce returnează un reprezentant al serviciului pe calculatorul clientului.

Astfel referinţa la serviciu se obţine prin

```
CmmdcWSService service=new CmmdcWSService();
CmmdcWS port=service.getCmmdcWSPort();
```

Prin variabila *port* putem apela orice operaţie a serviciului.

Codul clientului sincron este

```
1  package cmmdc.client;
2  import java.util.Scanner;

4  public class CmmdcClient {
5    public static void main(String[] args) {
6      try {
7        CmmdcWS port=new CmmdcWSService().getCmmdcWSPort();
8        Scanner scanner=new Scanner(System.in);
9        System.out.println("m=");
10       long m=scanner.nextLong();
11       System.out.println("n=");
12       long n=scanner.nextLong();
13       long result=port.cmmdc(m,n);
14       System.out.println("Cmmdc="+result);
```

```
15      }
16      catch (Exception e) {
17        System.out.printf("AnException : "+e.getMessage());
18      }
19    }
20  }
```

Alternativ, serviciul şi clientul se pot dezvolta în mediul de dezvoltare Netbeans. În acest caz, realizarea fişierelor de configurare nu mai este în sarcina programatorului. De asemenea, definirea operaţiilor oferite de serviciul Web se pot introduce prin intermediul unei interfeţe grafice, Netbeans generând antetul metodelor corespunzătoare.

Client Web al serviciului Web

Clientul Web este reprezentat de pagina JSP (*index.jsp*)

```
1  <html>
2       <body>
3       <form method="post">
4           <table>
5             <tr>
6               <td>Primul numar este </td>
7               <td><input type="text" name="m" size=5 value="1"> </td>
8             </tr>
9             <tr>
10              <td>Al doilea numar este </td>
11              <td><input type="text" name="n" size=5 value="1"> </td>
12            </tr>
13            <tr>
14              <td><input type="submit" value="Calculeaza"></td>
15              <td/>
16            </tr>
17          </table>
18      </form>
19      <%
20      try {
21        cmmdc.client.CmmdcWSService service=
22          new cmmdc.client.CmmdcWSService();
23        cmmdc.client.CmmdcWS port=service.getCmmdcWSPort();
24        String sm = request.getParameter("m");
25        String sn = request.getParameter("n");
26        long m=((sm==null)||("".equals(sm)))?1:Long.parseLong(sm);
27        long n=((sn==null)||("".equals(sn)))?1:Long.parseLong(sn);
28        long rez=port.cmmdc(m,n);
29        String result=(new Long(rez)).toString();
30        out.println("Result = "+result);
31      }
32      catch (Exception e) {
33        out.println("Exception : "+e.getMessage());
34      }
35      %>
36      <hr/>
37      </body>
38  </html>
```

Structura aplicaţiei Web care se desfăşoară în serverul Web este

```
client-web
|-->WEB-INF
|     |--> classes
|     |      |     *.class
|     |--> lib
|     |      web.xml
|     index.jsp
```

Fişierele `class` sunt cele generate de `wsimport` la dezvoltarea aplicaţiei. Fişierul *web.xml* este

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app version="2.4" xmlns="http://java.sun.com/xml/ns/j2ee"
      xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
      xsi:schemaLocation="http://java.sun.com/xml/ns/j2ee
      http://java.sun.com/xml/ns/j2ee/web-app_2_4.xsd">
         <session-config>
              <session-timeout>
                   30
              </session-timeout>
         </session-config>
         <welcome-file-list>
    <welcome-file>index.jsp</welcome-file>
  </welcome-file-list>
</web-app>
```

Valoarea *jspclient* din *sun-web.xml* reprezintă numele de apel al aplicaţiei client şi coincide cu numele arhivei `war`.

Clienţi *asincroni*

JAX-WS şi prin urmare şi *Metro* oferă posibilitatea dezvoltării de clienţi *asincroni*. Caracterul asincron se referă la faptul că programul client nu se blochează în aşteptarea răspunsului oferit de serviciul Web.

De această dată utilitarul `wsimport` se foloseşte indirect, mai precis prin intermediul clasei `com.sun.tools.ws.ant.WsImport` din Metro.

Generarea metodelor utilizate de client în acest scop se obţine incluzând proprietatea

`binding=`*"etc/custom-client.xml"*

în sarcina `wsimport`, iar conţinutul fişierului *custom-client.xml* este

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<bindings
     xmlns:xsd="http://www.w3.org/2001/XMLSchema"
     xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/"
     wsdlLocation="http://localhost:8080/jaxws-cmmdc/cmmdcws?wsdl"
     xmlns="http://java.sun.com/xml/ns/jaxws">
     <bindings node="wsdl:definitions">
          <enableAsyncMapping>true</enableAsyncMapping>
     </bindings>
</bindings>
```

Codul corespunzător din `build.xml` este

```
<taskdef name="wsimport" classname="com.sun.tools.ws.ant.WsImport">
      <classpath refid="myclasspath"/>
</taskdef>

<target name="generate-client" depends="init">
      <wsimport
        debug="true"
        verbose="${verbose}"
        keep="true"
        destdir="${build.dir}"
        package="${app.name}.client"
        binding="etc/custom-client.xml"
        xendorsed="true"
        wsdl="${wsdl.uri}">
        <arg line="-extension"/>
      </wsimport>
</target>
```

Pe calculatorul clientului, în clasa reprezentând serviciul (*CmmdcWS* - pentru exemplul tratat), pentru fiecare metodă a serviciului se generează una din metodele

- `public Response<`*Metoda*`Response>` *metoda*`Async(` *lista parametrilor formali*`)`
- `public Future<?>` *metoda*`Async(` *lista parametrilor formali,* `AsyncHandler<`*Metoda*`Response>` *asyncHandler*`)`

Pentru exemplul tratat *metoda=cmmdc.*

Cele două metode oferă posibilitatea construirii a câte unui program client specific, denumite, respectiv modelul *polling* şi modelul *callback.*

Au intervenit interfeţele

- `public interface Response<T> extends java.util.concurrent.Future<T>`

  Un obiect de acest tip conţine răspunsul serviciului. Răspunsul se obţine cu metoda `T get(),` moştenită de la `Future.`

  Metoda `boolean idDone()` returnează `true` dacă s-a primit răspuns.

- `public interface AsyncHandler<T>`

  Interfaţa declară metoda `void handleResponse(Response<T>` *res*`)` responsabilă de prelucrarea răspunsului.

Exemplul 3.2.1 *Client construit pe modelul polling.*

```
1  package cmmdc.client;
2  import javax.xml.ws.Response;
3  import java.util.Scanner;

5  public class CmmdcAsyncClient {
6    public static void main(String[] args) {
7      long delta=500;
8      try {
9        CmmdcWS port=new CmmdcWSService().getCmmdcWSPort();
10       Scanner scanner=new Scanner(System.in);
11       System.out.println("m=");
12       long m=scanner.nextLong();
13       System.out.println("n=");
14       long n=scanner.nextLong();

16       Response<CmmdcResponse> response=port.cmmdcAsync(m,n);
17       while(!response.isDone()){
18           System.out.println("Wait "+delta+" ms");
19           Thread.sleep(delta);
20       }
21       CmmdcResponse output=response.get();
22       long result=output.getReturn();
23       System.out.println("Cmmdc="+result);
24     }
25     catch (Exception e) {
26       System.out.printf("AnException : "+e.getMessage());
27     }
28   }
29 }
```

Exemplul 3.2.2 *Client construit pe modelul callback.*

```
1  package cmmdc.client;
2  import java.util.concurrent.ExecutionException;
3  import java.util.concurrent.Future;
4  import javax.xml.ws.Response;
5  import javax.xml.ws.AsyncHandler;
6  import java.util.Scanner;

8  public class CmmdcAsyncClient {
9    public static void main(String[] args) {
10     long delta=500;
11     try {
12       CmmdcWS port=new CmmdcWSService().getCmmdcWSPort();
13       Scanner scanner=new Scanner(System.in);
14       System.out.println("m=");
15       long m=scanner.nextLong();
16       System.out.println("n=");
17       long n=scanner.nextLong();

19       CmmdcAsyncHandler asyncHandler=new CmmdcAsyncHandler();
20       Future<?> response=port.cmmdcAsync(m,n,asyncHandler);
21       while(!response.isDone()){
22           System.out.println("Wait "+delta+" ms");
23           Thread.sleep(delta);
24       }
```

```
25        CmmdcResponse output=asyncHandler.getResponse();
26        long result=output.getReturn();
27        System.out.println("Cmmdc="+result);
28      }
29      catch (Exception e) {
30        System.out.printf("AnException : "+e.getMessage());
31      }
32    }
33  }

35  class CmmdcAsyncHandler implements AsyncHandler<CmmdcResponse>{
36    private CmmdcResponse output;

38    public void handleResponse(Response<CmmdcResponse> response){
39      try{
40        output=response.get();
41      }
42      catch(ExecutionException e){
43        System.out.println("ExecutionException : "+e.getMessage());
44        e.printStackTrace();
45      }
46      catch(InterruptedException e){
47        System.out.println("InterruptedException : "+e.getMessage());
48        e.printStackTrace();
49      }
50    }

52    CmmdcResponse getResponse(){
53      return output;
54    }
55  }
```

3.2.2 Componentă EJB sesiune stateless ca serviciu Web

Şablonul pentru crearea unei componente EJB de tip *stateless session* ca serviciu Web de tip JAX-WS este

```
import javax.ejb.Stateless;
import javax.jws.WebService;
import javax.jws.WebMethod;
import javax.jws.WebParam;

@WebService
@Stateless
public class Componenta{
  @WebMethod
  public tip metoda(@WebParam(name="numeVarFormala")tip numeVarFormala,. . .){. . .}
  . . .
}
```

Exemplul 3.2.3 *Cel mai mare divizor comun a două numere naturale*

Componentei EJB are codul

```
1  package cmmdcws;
2  import javax.jws.WebService;
3  import javax.jws.WebMethod;
4  import javax.jws.WebParam;
5  import javax.ejb.Stateless;

7  @WebService
8  @Stateless
9  public class CmmdcEJB {

11     @WebMethod
12     public long cmmdc(@WebParam(name = "m") long m,
13                       @WebParam(name = "n") long n){. . .}
14 }
```

Clasa compilată obişnuit se arhivează cu extensia `jar` şi se desfăsoară în serverul de aplicaţie (*glassfish*). Diferenţa faţă de varianta de serviciu ca servlet constă în structura arhivei care se desfăşoară în serverul Web.

```
cmmdc-ejb
|--> cmmdcws
|    |     CmmdcEJB.class
```

Clientul este unul obişnuit pentru serviciul Web de tip JAX-WS. În cazul de faţă codul clientului este

```
1  package client;
2  import cmmdcws.*;
3  import java.util.Scanner;

5  public class CmmdcClient {
6    public static void main(String[] args) {
7      try {
8        CmmdcEJB port=new CmmdcEJBService().getCmmdcEJBPort();
9        Scanner scanner=new Scanner(System.in);
10       System.out.println("m=");
11       long m=scanner.nextLong();
12       System.out.println("n=");
13       long n=scanner.nextLong();
14       long result=port.cmmdc(m,n);
15       System.out.println("Cmmdc="+result);
16     }
17     catch (Exception e) {
18       System.out.printf("AnException : "+e.getMessage());
19     }
20   }
21 }
```

Reamintim că dezvoltarea clientului se face cu serviciul Web activ, fiind necesară generarea unor clase - reprezentante ale serviciului - pe calculatorul clientului. La generarea claselor de către `wsimport,` referinţa URL a serviciului Web trebuie să corespundă serverului Web care conţine serviciul în timpul utilizării.

3.2.3 Servicii *jaxws* dezvoltate prin *maven*

Dezvoltarea aplicaţiei server

Serviciul `jaxws` se va desfăşura în serverul Web *glassfish.*
Vom dezvolta serviciul pentru calculul celui mai mare divizor comun cu

- contextul *jaxws-cmmdc*
- numele de apel *cmmdcws*

Aceşti parametrii sunt specificaţi în fişierul `sun-jaxws.xml`.
Dezvoltarea aplicaţiei constă din:

1. Generarea aplicaţiei

```
mvn archetype:create
   -DgroupId=cmmdc.server
   -DartifactId=jaxws-cmmdc
   -DarchetypeArtifactId=maven-archetype-webapp
```

2. Se adaptează structura de cataloage şi fişiere la

```
jaxws-cmmdc
|--> src
|    |--> main
|    |    |--> java
|    |    |    |--> cmmdc
|    |    |    |    |--> server
|    |    |    |    |      |     CmmdcWS.java
|    |    |--> resources
|    |    |--> webapp
|    |    |    |--> WEB-INF
|    |    |    |    |    web.xml
|    |    |    |    |    sun-jaxws.xml
|    pom.xml
```

3. Fişierul `pom.xml` se completează cu

```
<dependencies>
   . . .
   <dependency>
      <groupId>org.glassfish.metro</groupId>
      <artifactId>webservices-rt</artifactId>
      <version>2.3</version>
    </dependency>
. . .
</dependencies>
<repositories>
 <repository>
     <id>maven2-repository.java.net</id>
     <name>Java.net Repository for Maven 2</name>
     <url>http://download.java.net/maven/2/</url>
```

```
  </repository>
</repositories>
<pluginRepositories>
  <pluginRepository>
      <id>maven2-repository.java.net</id>
      <name>Java.net Repository for Maven 2</name>
      <url>http://download.java.net/maven/2/</url>
      <layout>default</layout>
  </pluginRepository>
</pluginRepositories>
```

4. Prelucrarea constă din

   (a) `mvn clean package`

   (b) Fişierul `war` care rezultă se desfăşoară în *glassfish.*

Dezvoltarea aplicaţiei client

Dezvoltarea aplicaţiei constă din:

1. Generarea aplicaţiei

```
mvn archetype:create
   -DgroupId=cmmdc.client
   -DartifactId=jaxws-cmmdc-client
   -DarchetypeArtifactId=maven-archetype-quickstart
```

2. Se adaptează structura de cataloage şi fişiere la

```
jaxws-cmmdc-client
|--> src
|    |--> main
|    |    |--> java
|    |    |    |--> cmmdc
|    |    |    |    |--> client
|    |    |    |    |      |     CmmdcClient.java

|    pom.xml
```

   În clasa *CmmdcClient.java* se introduce

```
import cmmdc.server.*;
```

3. Fişierul `pom.xml` se completează cu

```
  <dependencies>
     . . .
    <dependency>
       <groupId>com.sun.xml.ws</groupId>
       <artifactId>jaxws-rt</artifactId>
       <version>2.1</version>
```

```
 </dependency>
. . .
</dependencies>
<build>
  <plugins>
    <plugin>
      <groupId>org.jvnet.jax-ws-commons</groupId>
      <artifactId>jaxws-maven-plugin</artifactId>
      <version>2.2</version>
      <executions>
        <execution>
          <goals>
            <goal>wsimport</goal>
          </goals>
          <configuration>
            <wsdlUrls>
               <wsdlUrl>
                  http://localhost:8080/jaxws-cmmdc/cmmdcws?wsdl
               </wsdlUrl>
            </wsdlUrls>
            <packageName>cmmdc.client</packageName>
          </configuration>
          <phase>generate-sources</phase>
        </execution>
     </executions>
    </plugin>
 </plugins>
```

4. Prelucrarea constă din

   (a) `mvn clean compile`

   (b) `mvn exec:java -Dexec.mainClass="cmmdc.client.CmmdcClient"`

Capitolul 4

Servicii JAX-RS

4.1 Representational State Transfer

REpresentational State Transfer (REST) este un model de arhitectură de aplicaţie distribuită REST specifică modul cum o resursă - entitate care conţine informaţie specifică - este definită şi cum poate fi adresată.

Identificarea unei resurse se face printr-un URI (*Universal Resource Identifier*).

Transferul resursei către un client sau prelucrarea resursei se face utilizând o interfaţă care conţine o mulţime de operaţii `http: GET, POST, PUT, DELETE.`

Resursele sunt fără stare iar modelul de aplicaţie este cel de client-server.

Pentru sistemelele care satisfac aceste restricţii se utilizează terminologia *RESTful.*

Principalul exemplu de sistem RESTful este World Wide Web (WWW) cu protocolul Hyper Text Transfer Protocol (HTTP).

Serviciile Web bazate pe REST se bazează pe accesul la resurse - definite prin identificatori şi nu pe apelarea unor metode, ca în modelul Remote Procedure Call (RPC).

Standardul *JAX-RS Java API for RESTful Web Services* a ajuns la versiunea 2. (Versiunea 1 este definită de JSR 311, iar versiunea 2 este definită de JSR 339)

În prezent, în Java, există mai multe implementări pentru servicii REST.

Entităţile utilizate sunt:

- *Clasă resursă - Resource class.* Resursa Web este reprezintă de o clasă Java cu adnotări JAX-RS. *Clasa resursă rădăcină - Root resource class.*

Clasă cu adnotarea `@Path`. Resursele adiacente se definesc relativ la această clasă (resursă).

- *Metoda de identificare a cererii - Request method designator.* Adnotarea `@GET / @POST /@PUT / @DELETE` este folosită pentru identificarea cererii HTTP în vederea desemnării metodei de generare / prelucrare a resursei.
- *Metodă de generare / prelucrare a resusei - Resource method.*
- *Localizator a resurselor adiacente - Sub-resource locator.* Metodă pentru localizarea a resurselor adiacente, adică a resurselor care se specifică relativ la resursa rădăcină.
- *Metoda de generare / prelucrare a unei resurse adiacente - Sub-resource method.*
- *Provider* o implementare a interfeţei JAX-RS.

O implementare de referinţă (*Reference Implementation*) este oferită de pachetul *jersey-*.*.** realizat de *Oracle.*

4.2 *Jersey-2*

4.2.1 Generarea resurselor

Instalarea. Se va descărca o arhivă care conţine resursele necesare sub formă de fişiere `jar`.

Serverul Web care se va utiliza va fi *apache-tomcat.* În vederea desfăşurării, serviciul se arhivează cu extensia `war`. Numele arhivei va desemna numele serviciului. Structura care se arhivează va fi

```
catalogul_serviciului_RESTful
|--> WEB-INF
|    |--> classes
|    |    |--> resources
|    |    |    |   *.class
|    |--> lib
|    |    |    *.jar
|    |    web.xml
|    index.html sau index.jsp
```

Printre fişierele **.class* se găsesc resursele serviciului. *index.html* sau *index.jsp* oferă posibilitatea apelării serviciului. Alternativ un serviciu JAX-RS se poate apela dintr-un program client.
Fişierul `web.xml` este

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app version="2.5" xmlns="http://java.sun.com/xml/ns/javaee"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://java.sun.com/xml/ns/javaee
  http://java.sun.com/xml/ns/javaee/web-app_2_5.xsd">
    <servlet>
        <servlet-name>Jersey Web Application</servlet-name>
        <servlet-class>
            org.glassfish.jersey.servlet.ServletContainer
        </servlet-class>
        <init-param>
            <param-name>
                jersey.config.server.provider.packages
            </param-name>
            <param-value>resources</param-value>
        </init-param>
        <load-on-startup>1</load-on-startup>
    </servlet>
    <servlet-mapping>
        <servlet-name>Jersey Web Application</servlet-name>
        <url-pattern>/resources/*</url-pattern>
    </servlet-mapping>
</web-app>
```

Resursele jar necesare sunt

asm-all-repacked-*.jar	cglib-*.jar
guava-*.jar	hk2-api-*.jar
hk2-locator-*.jar	hk2-utils-*.jar
javax.annotation-api-*.jar	javax.inject-*.jar
javax.ws.rs-api-*.jar	jersey-client-*.jar
jersey-common-*.jar	jersey-container-servlet-core-*.jar
jersey-server-*.jar	osgi-resource-locator-*.jar
validation-api-*.jar	

Exemplul 4.2.1 *Serviciul RESTful Hello World care constă în furnizarea textului "Hello World".*

Codul sursă al resursei este:

```
package resources;

import javax.ws.rs.Produces;
import javax.ws.rs.Path;
import javax.ws.rs.GET;

@Path("helloworld")
public class HelloWorldResource {
```

```
10    public HelloWorldResource() {}

12    @GET
13    @Produces("text/plain")
14    public String getText() {
15      return ("Hello World");
16    }

18    @Path("html")
19    @GET
20    @Produces("text/html")
21    public String getAsHtml() {
22      return "<html><head></head><body bgcolor=\"#bbeebb\"><center>
23        <p>Hello World</p></center></body></html>";
24    }

26    @Path("xml")
27    @GET
28    @Produces("application/xml")
29    public String getAsXml() {
30        return "<response>Hello World</response>";
31    }
32  }
```

Adnotarea `@Path("helloworld")` face ca localizarea resursei să fie

`http://`*host:port/NumeServiciuRESTful*`/resources/helloworld`

Metoda *getText()* este o metodă de generare a resursei şi are adnotările

- `@GET` - răspunde la o cerere GET;
- `@Produces(String mime-type)` - rezultatul produs are tipul MIME (Multipurpose Internet Mail Exchange) "text/plain".

Suplimentar sunt definite două resurse adiacente. În urma adnotării `@Path("html")`, localizarea resursei este

`http://`*host:port/NumeServiciuRESTful*`/resources/helloworld/html`

La o solicitare GET, resursa (clasa) furnizează clientului o pagină `html`.

Analog se tratează resursa adiacentă cu adnotarea `@Path ("xml")`.

Apelarea serviciului prin internediul unui navigator se poate face prin intermediul fişierului *index.html*

```
1  <html>
2    <body>
3    <table border="2">
4      <tr>
5        <td>
6          <a href="/HelloWorld/resources/helloworld">
7            Rezultat "text/plain"</a>
8        </td>
```

```
9       </tr>
10      <tr>
11        <td>
12          <a href="/HelloWorld/resources/helloworld/html">
13            Rezultat "text/html"</a>
14        </td>
15      </tr>
16      <tr>
17        <td>
18          <a href="/HelloWorld/resources/helloworld/xml">
19            Rezultat "application/xml"</a>
20        </td>
21      </tr>
22      </table>
23      </body>
24  </html>
```

În acest caz, apelarea este `http://`*host:port/HelloWorld.*

Elaborarea unui program client pentru un serviciu RESTful realizat prin *jersey* se poate baza pe

1. Clasa `java.net.HttpURLConnection` sau `java.net.ssl.HttpsURLConnection` din distribuţia jdk.

   Conexiunea cu resursa furnizată de serviciul RESTful se obţine prin

   ```
   URL url=new URL(String url_resursa);
   HttpURLConnection conn=(HttpURLConnection) url.openConnection();
   ```

   Prin metodele clasei `HttpURLConnection` se determină

   - Codul reîntors de serviciul RESTful - `public int getResponseCode()`. Succesul apelului este dat de codul 200.
   - Mesajul explicativ al codului - `public String getResponseMessage()`.
   - Fluxul care furnizează resursa - `public InputStream getInputStream() throws IOException`.
   - În final trebuie deconectată conexiunea prin `void disconnect()`.

   Un client al serviciului *RESTfull* anterior este clasa

```
1   import java.net.URL;
2   import java.net.HttpURLConnection;
3   import java.io.IOException;
4   import java.io.InputStreamReader;
5   import java.io.BufferedReader;

7   public class Client{
8     public static void main(String args[]){
9       System.out.println("Rezultat / \'text/plain\'");
10      String urlStr=
```

```
11        "http://localhost:8080/HelloWorld/resources/helloworld";
12      try{
13        String rezultat=httpGetText(urlStr);
14        System.out.println(rezultat);
15      }
16      catch(Exception e){
17        System.out.println("Exception message : "+e.getMessage());
18      }
19      System.out.println();

21      System.out.println("Rezultat / \'text/html\'");
22      urlStr =
23        "http://localhost:8080/HelloWorld/resources/helloworld/html";
24      try{
25        String rezultat=httpGetText(urlStr);
26        System.out.println(rezultat);
27      }
28      catch(Exception e){
29        System.out.println("Exception message : "+e.getMessage());
30      }
31      System.out.println();

33      System.out.println("Rezultat / \'application/xml\'");
34      urlStr =
35        "http://localhost:8080/HelloWorld/resources/helloworld/xml";
36      try{
37        String rezultat=httpGetText(urlStr);
38        System.out.println(rezultat);
39      }
40      catch(Exception e){
41        System.out.println("Exception message : "+e.getMessage());
42      }
43    }

45    public static String httpGetText(String urlStr) throws IOException {
46      URL url=new URL(urlStr);
47      HttpURLConnection conn=(HttpURLConnection) url.openConnection();

49      if (conn.getResponseCode() != 200) {
50        throw new IOException(conn.getResponseMessage());
51      }

53      // Buffer the result into a string
54      BufferedReader rd = new BufferedReader(
55          new InputStreamReader(conn.getInputStream()));
56      StringBuilder sb = new StringBuilder();
57      String line;
58      while ((line = rd.readLine()) != null) {
59        sb.append(line+"\n");
60      }
61      rd.close();

63      conn.disconnect();
64      return sb.toString();
65    }
66  }
```

2. Resursele interfeţei de programare (API) *Jersey Client,* conţinute în pa-

chetul `com.sun.jersey.api.client`. Această varianta asigură o prelucrere mai simplă şi unitară a rezultatelor furnizate de un serviciu RESTful.

Şablonul de prelucrare constă din:

(a) Crearea unei instanţe a clasei `javax.ws.rs.client.Client`

```
Client client = ClientBuilder.newClient();
```

(b) Fixarea adresei serviciului RESTful în interfaţa `javax.ws.rs.client.WebTarget`.

```
WebTarget webTarget=client.target(String url_resursa);
```

(c) Utilizarea metodelor clasei `WebTarget`:

- `Invocation.Builder request()`
  `Invocation.Builder request(String...` *acceptedResponseTypes*`)`
  `Invocation.Builder request(MediaType...` *acceptedResponseTypes*`)`
  Construieşte obiectul care execută apelarea resursei.
- `URI getUri()`
  Furnizează adresa resursei.
- `WebTarget queryParam(String` *name,* `Object...` *values*`)`
  Adaugă parametrii cererii.

(d) Apelarea resurselor prin metodele interfeţei `javax.ws.rs.client.Invocation.Builder`

- `<T> T get(Class<T>` *responseType*`)`
- `<T> T post(Class<T>` *responseType*`)`
- `<T> T post(Entity<?>` *entity,* `Class<T>` *responseType*`)`

Cu această tehnologie, codul sursă al clasei client este

```
1  package hw;
2  import javax.ws.rs.client.Client;
3  import javax.ws.rs.client.ClientBuilder;
4  import javax.ws.rs.client.WebTarget;
5  import javax.ws.rs.client.Invocation;
6  import java.util.Scanner;
7  public class JerseyClient {
8    public static void main(String args[]) {
9      Client client = ClientBuilder.newClient();

11     Scanner scanner=new Scanner(System.in);
12     System.out.println("Introduceti tipul raspunsului : ");
13     System.out.println("( plain / html / xml )");
```

```
14      String tip=scanner.next().trim();
15      String response="";
16      WebTarget webTarget=null;
17      Invocation.Builder invocationBuilder=null;
18      switch(tip){
19        case "plain":
20          webTarget=client.target(
21           "http://localhost:8080/HelloWorld/resources/helloworld");
22          invocationBuilder=webTarget.request();
23          response=invocationBuilder.get(String.class);
24          break;
25        case "html":
26          webTarget=client.target(
27           "http://localhost:8080/HelloWorld/resources/helloworld/html");
28          invocationBuilder=webTarget.request();
29          response=invocationBuilder.get(String.class);
30          break;
31        case "xml":
32          webTarget=client.target(
33           "http://localhost:8080/HelloWorld/resources/helloworld/xml");
34          invocationBuilder=webTarget.request();
35          response=invocationBuilder.get(String.class);
36          break;
37      }
38      System.out.println(response);
39    }
40  }
```

Dacă catalogul resurselor (*resources* în cazul exemplului anterior) se doreşte ascuns, atunci se poate înlocui cu o referinţă virtuală, de exemplu *webresources*. În acest caz codul de apelare va fi

http://localhost:8080/HelloWorld/webresources/helloworld.

Aceast efect se obţine introducănd clasa

```
1  package resources;
2  import java.util.Set;
3  import javax.ws.rs.ApplicationPath;
4  import javax.ws.rs.core.Application;

6  @ApplicationPath("webresources")
7  public class MyApplication extends Application {

9    @Override
10   public Set<Class<?>> getClasses() {
11     Set<Class<?>> resources = new java.util.HashSet<>();
12     resources.add(HelloWorldResource.class);
13     return resources;
14   }
15 }
```

care leagă referinţa virtuală *webresources* de clasa care generează resursa rădăcină, *HelloWorldResource* . Clasa `javax.ws.rs.core.Application` conţine metodele

- `public Set<Class<?>> getClasses()`

- `public Set<Class<?>> getSingletons()`

Prezenţa arhivei `jersey-container-servlet-*.jar` permite eliminarea fişierului `web.xml`.

Fiecare tip de resursă a fost generat ca o subresursă cu o cale specifică. Această abordare a făcut posibilă realizarea unui client Web şi a unui client bazat pe clasa `java.net.URL`. Un client *jersey* poate prelua o resursă pe baza tipului MIME (*Multipurpose Internet Mail Extensions*). Modificăm codul clasei *HelloWorldResource* în

```
1  package resources;
2
3  import javax.ws.rs.Produces;
4  import javax.ws.rs.Path;
5  import javax.ws.rs.GET;
6  import javax.ws.rs.core.MediaType;
7
8  @Path("helloworld")
9  public class HelloWorldResource {
10
11   public HelloWorldResource() {}
12
13   @GET
14   @Produces(MediaType.TEXT_PLAIN)
15   public String getText() {
16     return ("Hello World");
17   }
18
19   @GET
20   @Produces(MediaType.TEXT_HTML)
21   public String getAsHtml() {
22     return "<html><head></head><body bgcolor=\"#bbeebb\"><center>
23       <p>Hello World</p></center></body></html>";
24   }
25
26   @GET
27   @Produces(MediaType.TEXT_XML)
28   public String getAsXml() {
29       return "<response>Hello World</response>";
30   }
31
32   @GET
33   @Produces(MediaType.APPLICATION_XML)
34   public String getAsAppXml() {
35       return "<response>Hello World</response>";
36   }
37
38   @GET
39   @Produces(MediaType.APPLICATION_JSON)
40   public String getAsJson() {
41       return "[\"Hello World\"]";
42   }
43 }
```

iar codul clasei client *JerseyClient* în

```
1  package hw;
2  import javax.ws.rs.client.Client;
3  import javax.ws.rs.client.ClientBuilder;
4  import javax.ws.rs.client.WebTarget;
5  import javax.ws.rs.client.Invocation;
6  import javax.ws.rs.core.MediaType;

8  public class JerseyClient {
9    public static void main(String args[]) {
10     Client client = ClientBuilder.newClient();
11     String response="";
12     WebTarget webTarget=client.target
13       "http://localhost:8080/HelloWorld/resources/helloworld");
14     Invocation.Builder invocationBuilder=
15       webTarget.request(MediaType.TEXT_PLAIN_TYPE);
16     response=invocationBuilder.get(String.class);
17     System.out.println(response);
18     System.out.println();
19     invocationBuilder=webTarget.request(MediaType.TEXT_HTML_TYPE);
20     response=invocationBuilder.get(String.class);
21     System.out.println(response);
22     System.out.println();
23     invocationBuilder=webTarget.request(MediaType.TEXT_XML_TYPE);
24     response=invocationBuilder.get(String.class);
25     System.out.println(response);
26     System.out.println();
27   }
28 }
```

În exemplul anterior, codul `html` a fost generat de o metodă. Dacă se oferă un fişier `html` existent, atunci acesta se furnizează prin

```
import javax.servlet.ServletContext;
. . .
@Context ServletContext sc;
  //@Path("html")
  @ProduceMime("text/html")
  @GET
  public InputStream doGet() {
    return sc.getResourceAsStream("/Index.html");
  }
```

Caracterul / din argumentul metodei `getResourceAsStream` precizează rădăcina aplicaţiei, mai precis catalogul aplicaţiei din `webapps`.

Printr-o adnotare `@Context` se pot defini variabile de tip `UriInfo, Request, HttpHeaders, ServletContext,` etc.

Exemplu de furnizare a unei imagini *jpg, png,* aflat în catalogul *resources* este dată de metoda

```
1  package resources;
2  import java.net.URL;
3  import javax.ws.rs.Produces;
4  import javax.ws.rs.Path;
5  import javax.ws.rs.GET;
6  import javax.activation.DataSource;
```

```
7  import javax.activation.FileDataSource;

9  @Path("imagine")
10 public class ImageResource {

12   public ImageResource() {}

14   @GET
15   @Produces("image/jpg")
16   public DataSource getImageRep() {
17         URL jpgURL = this.getClass().getResource("forest.jpg");
18         return new FileDataSource(jpgURL.getFile());
19   }
20 }
```

Pe partea de client, imaginea se obţine cu metoda

```
import java.awt.Image;
import java.awt.Toolkit;
. . .
public static Image httpGetImage(String urlStr) throws IOException {
  URL url=new URL(urlStr);
  return Toolkit.getDefaultToolkit().getImage(url);
}
```

unde *urlStr* localizează resursa.

4.2.2 Preluarea parametrilor

Există mai multe metode de programare a preluării argumentelor provenind din formulare html sau din programe client şi generarea resursei răspuns:

- Prin context;
- Prin adnotarea `@QueryParam` — `@FormParam`.

Parametrii necesari pentru generarea unei resurse pot fi transmişi sub forma unei liste plasat în urma adresei URL a serviciului RESTful. Argumentele sunt separate prin virgulă iar separatorul dintre URL şi lista argumentelor este /. În serviciul RESTful va apare adnotarea `@PathParam`.

Preluarea parametrilor prin context

Preluarea unui parametru implică utilizarea mai multor clase ale interfeţei JAX-RS.

- Interfaţa `javax.ws.rs.core.UriInfo`

  Metode

  - `MultivaluedMap<java.lang.String,java.lang.String> getQueryParameters()`
    Returnează lista parametrilor corespunzători cererii.

- Interfaţa `javax.ws.rs.core.MultivaluedMap<K,V> extends java.util.Map<K,java.util.List<V>>`

  Metode

  - `V getFirst(K` *key*`)`
    Returnează valoarea corespunzătoare cheii *key*.

- Clasa `javax.ws.rs.core.Response`

  Metode

  - `public static Response.ResponseBuilder ok(Object` *entity,* `String` *mimetype*`)`
    Crează un obiect care conţine reprezentarea răspunsului.

- Clasa `javax.ws.rs.core.Response.ResponseBuilder`

  Metode

  - `public abstract Response build()`
    Crează obiectul `Response` în urma executării metodei `ok`.

Pentru acelaşi exemplu codul serviciului este:

```
1  package resources;
2  import javax.ws.rs.core.MultivaluedMap;
3  import javax.ws.rs.core.Response;
4  import javax.ws.rs.core.Context;
5  import javax.ws.rs.core.UriInfo;
6  import javax.ws.rs.Path;
7  import javax.ws.rs.GET;
8  import java.net.URLDecoder;

10 @Path("hello")
11 public class HelloResource {

13   public HelloResource() {}

15   @Context UriInfo uriInfo;
16   @GET
17   public Response doGet(){
18     MultivaluedMap<String,String> params=uriInfo.getQueryParameters();
19     String nume=params.getFirst("nume");
20     String tip=null;
21     try{
22       tip=URLDecoder.decode(params.getFirst("tip"),"UTF-8");
```

```
23      }
24      catch(Exception e){
25          System.out.println(e.getMessage());
26      }
27      System.out.println("Param : "+nume+" "+tip);
28      Response r=null;
29      switch(tip){
30        case "text/plain":
31          r=Response.ok(getPlainRep(nume),"text/plain").build();
32          break;
33        case "text/html":
34          r=Response.ok(getHtmlRep(nume),"text/html").build();
35          break;
36      }
37      return r;
38    }

40    public String getPlainRep(String nume) {
41       return "Hello "+nume+" !";
42    }

44    public String getHtmlRep(String nume) {
45      return "<html><head></head><body bgcolor=\"#aaeeaa\"><center>
46        <h1>Hello "+nume+" ! </h1></center></body></html>";
47    }
48  }
```

Preluarea argumentelor prin adnotarea `@QueryParam` | `@FormParam`

Utilizarea adnotărilor `@QueryParam, @FormParam` permite preluarea parametrilor unui formular transmişi prin metoda GET, respectiv POST. În acest caz serverul Web injecteaă parametrii metodelor în care este prezentă adnotarea.

Exemplul 4.2.2 *Serviciu RESTful care furnizează mesajul "Hello nume", unde "nume" este un parametru al clientului. Un al doilea parametru "tip" fixează natura raspunsului* `text/plain` *sau* `text/html`.

Codul serviciului cu metoda GET este:

```
1  package resources;
2  import javax.ws.rs.Path;
3  import javax.ws.rs.GET;
4  import javax.ws.rs.QueryParam;
5  import javax.ws.rs.core.Response;
6  import java.net.URLDecoder;

8  @Path("hello")
9  public class HelloResource {

11   public HelloResource() {}

13   @GET
```

```
14   public Response processQuery(
15       @QueryParam("nume") String nume,
16       @QueryParam("tip") String tip) {
17     String tip0=null;
18     try{
19       tip0=URLDecoder.decode(tip.trim(),"UTF-8");
20     }
21     catch(Exception e){
22        System.out.println(e.getMessage());
23     }
24     Response r=null;
25     switch(tip0){
26       case "text/plain":
27         r=Response.ok(getPlainRep(nume),"text/plain").build();
28         break;
29       case "text/html":
30         r=Response.ok(getHtmlRep(nume),"text/html").build();
31         break;
32     }
33     return r;
34   }

36   public String getPlainRep(String nume) {
37      return "Hello "+nume+" !";
38   }

40   public String getHtmlRep(String nume) {
41     return "<html><head></head><body bgcolor=\"#aaeeaa\"><center>
42      <h1>Hello "+nume+" ! </h1></center></body></html>";
43   }
44 }
```

Apelarea serviciului *RESTful* dintr-un navigator se obţine cu

```
1  <html>
2    <body>
3      <center>
4        <h1> Serviciu Hello de tip RESTful </h1>
5        <form method="get"
6              action="/Hello/resources/hello">
7         <p> Introduceti numele </p>
8         <input type="text" name="nume" size="10" />
9         <p> Selectati tipul raspunsului </p>
10        <select name="tip">
11          <option value="text/plain"> Text/Plain </option>
12          <option value="text/html"> Text/Html </option>
13        </select>
14        <p>
15        <input type="submit" value="Apeleaza" />
16       </form>
17     </center>
18     </body>
19 </html>
```

Utilizând metoda POST în programul serviciului, adnotarea `@QueryParam` se înlocuieşte cu adnotarea `@FormParam`.

În cazul unui serviciu *RESTful* care răspunde la o cerere GET, în programul client bazat pe clasa `HttpURLConnect`, datele de intrare trebuie codate *UTF-8*

şi asamblate în adresa URL:

```
import java.net.URL;
import java.net.HttpURLConnection;
import java.net.URLEncoder;
import java.io.IOException;
import java.io.InputStreamReader;
import java.io.BufferedReader;
import java.util.Scanner;

public class Client{
  public static void main(String args[]){
    Scanner scanner=new Scanner(System.in);
    try{
      String param="?";
      System.out.println("Numele");
      String nume=URLEncoder.encode(scanner.next(),"UTF-8");
      param=param+"nume="+nume+"&";
      param=param+"tip="+URLEncoder.encode("text/plain","UTF-8");
      String urlStr="http://localhost:8080/Hello/resources/hello"+param;
      System.out.println(urlStr);
      String rezultat=httpGetText(urlStr);
      System.out.println(rezultat);
    }
    catch(Exception e){
      System.out.println("Exception message : "+e.getMessage());
    }
    System.out.println();
  }

  public static String httpGetText(String urlStr) throws IOException{. . .}
}
```

În varianta *JerseyClient* codul clasei client este

```
package hello;
import javax.ws.rs.client.Client;
import javax.ws.rs.client.ClientBuilder;
import javax.ws.rs.client.WebTarget;
import java.util.Scanner;

public class JerseyClient {
  public static void main(String args[]) {
    Client client = ClientBuilder.newClient();
    WebTarget webTarget=
      client.target("http://localhost:8080/Hello/resources/hello");
    Scanner scanner=new Scanner(System.in);
    System.out.println("Introduceti tipul raspunsului : ");
    System.out.println("( plain / html )");
    String tip=scanner.next();
    System.out.println("Introduceti numele : ");
    String nume=scanner.next();
    String response=webTarget.queryParam("nume",nume).
          queryParam("tip","text/"+tip).request().get(String.class);
    System.out.println(response);
  }
}
```

Pentru metoda POST, în varianta simplă a clientului are codul

```
1  import java.net.URL;
2  import java.net.HttpURLConnection;
3  import java.net.URLEncoder;
4  import java.io.IOException;
5  import java.io.InputStreamReader;
6  import java.io.BufferedReader;
7  import java.io.PrintWriter;
8  import java.util.Scanner;

10 public class Client{
11   public static void main(String args[]){
12     Scanner scanner=new Scanner(System.in);
13     try{
14       System.out.println("Numele");
15       String nume=URLEncoder.encode(scanner.next(),"UTF-8");
16       String tip=URLEncoder.encode("text/plain","UTF-8");
17       String urlStr="http://localhost:8080/Hello/resources/hello";
18       String param="nume="+nume+"&tip="+tip;
19       System.out.println(param);
20       String rezultat=httpGetText(urlStr,param);
21       System.out.println(rezultat);
22     }
23     catch(Exception e){
24       e.printStackTrace();
25       System.out.println("ExceptionMessage "+e.getMessage());
26     }
27     System.out.println();
28   }

30   public static String httpGetText(String urlStr,String param)throws IOException{
31     URL url=new URL(urlStr);
32     HttpURLConnection conn=(HttpURLConnection) url.openConnection();
33     conn.setRequestMethod("POST");
34     conn.setUseCaches(false);
35     conn.setDoInput(true);
36     conn.setDoOutput(true);
37     conn.setRequestProperty("Content-Type","application/x-www-form-urlencoded");
38     PrintWriter pw=new PrintWriter(conn.getOutputStream());
39     pw.println(param);
40     pw.flush();
41     pw.close();
42     if (conn.getResponseCode() != 200) {
43       throw new IOException(conn.getResponseMessage());
44     }
45     BufferedReader rd = new BufferedReader(
46       new InputStreamReader(conn.getInputStream()));
47     StringBuilder sb = new StringBuilder();
48     String line;
49     while ((line = rd.readLine()) != null) {
50       sb.append(line+"\n");
51     }
52     rd.close();
53     conn.disconnect();
54     return sb.toString();
55   }
56 }
```

şi în varianta *JerseyClient* clasei client este

```
1  package hello;
2  import javax.ws.rs.client.Client;
3  import javax.ws.rs.client.Entity;
4  import javax.ws.rs.client.ClientBuilder;
5  import javax.ws.rs.client.WebTarget;
6  import javax.ws.rs.core.Form;
7  import javax.ws.rs.core.Response;
8  import java.util.Scanner;

10 public class JerseyClient {
11   public static void main(String args[]) {
12     Client client = ClientBuilder.newClient();
13     WebTarget webTarget=
14        client.target("http://localhost:8080/Hello/resources/hello");
15     Scanner scanner=new Scanner(System.in);
16     System.out.println("Introduceti tipul raspunsului : ");
17     System.out.println("( plain | html )");
18     String tip=scanner.next();
19     System.out.println("Introduceti numele : ");
20     String nume=scanner.next();
21     Form f=new Form().param("nume",nume).param("tip","text/"+tip);
22     Response response=webTarget.request().post(Entity.form(f));
23     String r = response.readEntity(String.class);
24     System.out.println(r);
25   }
26 }
```

Adnotarea `@PathParam`

În cazul utilizării metodei `http GET` o soluţie mai simplă pentru fixarea unui număr mic de parametri este introducerea lor în adnotarea `@Path` sub forma

```
@Path("refCale/{param1},{param2},. . .")
```

Metoda care preia parametri utilizează adnotarea `@PathParam`:

```
public T getResursa(@PathParam("param1") String p1,
                    @PathParam("param2") String p2,. . .){. . .}
```

Stringul care fixează URL-ul de apelare va fi de forma

.../resources/refCale/valParam1,valParam2

Exemplul 4.2.3 *Serviciu RESTful pentru calculul celui mai mare divizor comun a două numere naturale.*

Codul serviciului este

```
1  package resources;
2  import javax.ws.rs.Path;
3  import javax.ws.rs.PathParam;
4  import javax.ws.rs.Produces;
5  import javax.ws.rs.GET;
6  import javax.ws.rs.core.MediaType;

8  @Path("cmmdc/{num1},{num2}")
9  public class CmmdcResource {

11   public CmmdcResource(){}

13   @GET
14   @Produces(MediaType.APPLICATION_XML)
15   public String getCmmdcAsXML(@PathParam("num1") String sm,
16                               @PathParam("num2") String sn){
17     System.out.println(sm+" "+sn);
18     long m=Long.parseLong(sm);
19     long n=Long.parseLong(sn);
20     long c=cmmdc(m,n);
21     String result=(new Long(c)).toString();
22     return "<?xml version=\"1.0\" encoding=\"UTF-8\"?><rezultat>"+
23        result+"</rezultat>";
24   }

26   private long cmmdc(long m,long n) {. . .}
27 }
```

iar al clientului

```
1  import com.sun.jersey.api.client.Client;
2  import com.sun.jersey.api.client.WebResource;
3  import javax.ws.rs.core.MediaType;
4  import java.util.Scanner;

6  public class JerseyClient {
7    public static void main(String args[]) {
8      Scanner scanner=new Scanner(System.in);
9      System.out.println("Primul numar : ");
10     long m=scanner.nextLong();
11     System.out.println("Al doilea numar : ");
12     long n=scanner.nextLong();
13     String sm=(new Long(m)).toString();
14     String sn=(new Long(n)).toString();
15     Client client = Client.create();
16     WebResource webResource = client.resource(
17        "http://localhost:8080/cmmdcapp/resources/cmmdc/"+sm+","+sn);
18     String response=
19       webResource.accept(MediaType.APPLICATION_XML).get(String.class);
20     System.out.println();
21     System.out.println(MediaType.APPLICATION_XML);
22     System.out.println(response);
23   }
24 }
```

În acest caz un client Web se obţine cu AJAX

```
1  html>
```

```
<head>
<script type="text/javascript" >
<!--
function initRequest() {
    if (window.XMLHttpRequest) {
        return new XMLHttpRequest();
    } else if (window.ActiveXObject){
        return new ActiveXObject("Microsoft.XMLHTTP");
    }
}

function compute() {
    var mField=document.getElementById("m");
    var nField=document.getElementById("n");
    var url = "http://localhost:8080/cmmdcapp/resources/cmmdc/" +
        escape(mField.value)+","+escape(nField.value);
    var req = initRequest();
    req.onreadystatechange = function() {
            if (req.readyState == 4) {
                if (req.status == 200) {
                    parseMessages(req.responseXML);
                } else {
                    alert(req.status+" : "+req.statusText);
                }
            }
        };
    req.open("get", url, true);
    req.send(null);
}

function parseMessages(responseXML) {
  var r = responseXML.getElementsByTagName("rezultat")[0];
  var cmmdc=r.childNodes[0].nodeValue;
  document.getElementById("rezultat").innerHTML="Cmmdc = "+cmmdc;
}
-->
</script>

<title> Cmmdc AJAX</title>
</head>
 <body>

 <h1>Cmmdc with AJAX</h1>

 <p>
 Primul numar :
 <input type="text" id="m" value="1" size="15" >
 <p>
 Al doilea numar :
 <input type="text"  id="n" value="1" size="15" >
 <p>
 <input type="button" value="Calculeaza" onClick="compute()" >
 <p>
 Cel mai mare divizor comun al celor doua numere este
 <p>
 <div id="rezultat" />
 </body>
</html>
```

4.2.3 Date prin componentă Java

Datele către serviciul RESTful cât şi rezultatele pot fi trimise respectiv recepţionate printr-o componentă Java (bean). În acest caz se interpune un servlet pe rol de client *Jersey*. Componenta Java este acoperită de un obiect `javax.ws.rs.client.Entity`, iar tipul datelor este `MediaType.APPLICATION_JSON`. Tehnologia utilizată pentru conversia datelor are denumirea *MOXy POJO based JSON binding.*

Aplicaţiile client conlucrează cu servlet-ul.

- Clasa `javax.ws.rs.client.Entity<T>`

  Metode

  - `public static <T> Entity<T> entity(T` *entity,* `MediaType` *media Type*`)`

Exemplul 4.2.4

- Clasa componentei Java (POJO) - *CmmdcBean.java.*

```
1  package resources;
2
3  public class CmmdcBean{
4    private String sm;
5    private String sn;
6    private String result;
7
8    public void setSm(String sm){
9      this.sm=sm;
10   }
11   public String getSm(){
12     return sm;
13   }
14
15   public void setSn(String sn){
16     this.sn=sn;
17   }
18   public String getSn(){
19     return sn;
20   }
21
22   public void setResult(String result){
23     this.result=result;
24   }
25   public String getResult(){
26     return result;
27   }
28 }
```

- Clasa serviciului RESTful - *CmmdcResource.java.*

```
1  package resources;
2  import javax.ws.rs.Consumes;
3  import javax.ws.rs.POST;
4  import javax.ws.rs.Path;
5  import javax.ws.rs.Produces;
6  import javax.ws.rs.core.MediaType;

8  @Path("cmmdc")
9  public class CmmdcResource {
10   public CmmdcResource(){}

12   @POST
13   @Produces(MediaType.APPLICATION_JSON)
14   @Consumes(MediaType.APPLICATION_JSON)
15   public CmmdcBean myJob(CmmdcBean obj){
16     String sm=obj.getSm();
17     String sn=obj.getSn();
18     long m=Long.parseLong(sm);
19     long n=Long.parseLong(sn);
20     long c=cmmdc(m,n);
21     String cmmdc=(new Long(c)).toString();
22     obj.setResult(cmmdc);
23     return obj;
24   }

26   public long cmmdc(long m,long n) {. . .}
27 }
```

- Servlet-ul - *TestServlet.java*.

```
1  package resources;

3  import java.io.IOException;
4  import java.io.PrintWriter;
5  import javax.servlet.ServletException;
6  import javax.servlet.annotation.WebServlet;
7  import javax.servlet.http.HttpServlet;
8  import javax.servlet.http.HttpServletRequest;
9  import javax.servlet.http.HttpServletResponse;
10 import javax.ws.rs.client.Client;
11 import javax.ws.rs.client.ClientBuilder;
12 import javax.ws.rs.client.Entity;
13 import javax.ws.rs.client.WebTarget;
14 import javax.ws.rs.core.MediaType;

16 @WebServlet(urlPatterns = {"/TestServlet"})
17 public class TestServlet extends HttpServlet {

19   @Override
20   protected void doGet(HttpServletRequest req,
21      HttpServletResponse res)throws ServletException,IOException{
22     String sm=req.getParameter("m");
23     String sn=req.getParameter("n");
24     String tip=req.getParameter("tip");
25     System.out.println(sm+" : "+sn+" : "+tip);
26     Client client = ClientBuilder.newClient();
27     WebTarget target = client.target("http://"
28             + req.getServerName()
```

```
29              + ":"
30              + req.getServerPort()
31              + req.getContextPath()
32              + "/resources/cmmdc");
33      CmmdcBean bean=new CmmdcBean();
34      bean.setSm(sm);
35      bean.setSn(sn);
36      CmmdcBean obj = target
37              .request()
38              .post(Entity.entity(bean, MediaType.APPLICATION_JSON),
39                 CmmdcBean.class);
40      PrintWriter out = res.getWriter();
41      if(tip.equals("text/html")){
42        res.setContentType("text/html;charset=UTF-8");
43        out.println("<html>");
44        out.println("<head>");
45        out.println("<title>Servlet TestServlet</title>");
46        out.println("</head>");
47        out.println("<body>");
48        out.println("<h1>Servlet TestServlet at " +
49                           req.getContextPath() + "</h1>");
50        out.println("Received res: " + obj.getResult() + "<br><br>");
51        out.println("</body>");
52        out.println("</html>");
53      }
54      else{
55        res.setContentType("text/plain;charset=UTF-8");
56        out.println(obj.getResult());
57      }
58    }

60    @Override
61    protected void doPost(HttpServletRequest req,
62      HttpServletResponse res) throws ServletException, IOException{
63      doGet(req, res);
64    }
65  }
```

4.2.4 *Jersey* în *glassfish*

Pachetul *jersey* este conţinut în *glassfish.* În consecinţă arhiva `war` destinată desfăşurării unei aplicaţii nu trebuie să conţină resursele care ţin de *jersey.*

Aplicaţiile dezvoltate anterior functionează fără nici o modificare în *glassfish.*

Semnalăm arhitectura unei aplicaţii care utilizează o componentă EJB de tip *session stateless,* care este injectată în clasa serviciului RESTful. Injectarea se poate programa utilizând:

- adnotările `javax.annotation.ManagedBean` şi `javax.inject.Inject`.

  În acest caz, este nevoie de prezenţa unui fişier *beans.xml,* având codul

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://java.sun.com/xml/ns/javaee"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://java.sun.com/xml/ns/javaee
    http://java.sun.com/xml/ns/javaee/beans_1_0.xsd">
</beans>
```

- adnotarea `javax.ejb.EJB`.

Exemplul 4.2.5 *Calculul celui mai mare divizor comun a două numere naturale.*

Structura aplicaţiei este

```
Cmmdc
|--> WEB-INF
|      |--> classes
|      |      |--> cmmdc
|      |      |      |     App.class
|      |      |--> resources
|      |      |      |     CmmdcQueryResources.class
|      |      beans.xml
|      |      web.xml
|      index.html
```

Codul clasei *CmmdcQueryResources* este

```
package resources;
import javax.ws.rs.QueryParam;
import javax.servlet.ServletContext;
import javax.ws.rs.core.Response;
import javax.ws.rs.Produces;
import javax.ws.rs.Path;
import javax.ws.rs.GET;
import java.net.URLDecoder;
import javax.annotation.ManagedBean;
import javax.inject.Inject;
import cmmdc.App;

@Path("cmmdcquery")
@ManagedBean
public class CmmdcQueryResource {
  @Inject
  App obj;

  @GET
  public Response processQuery(
      @QueryParam("m") String sm,
      @QueryParam("n") String sn,
      @QueryParam("tip") String tip) {
    String tip0=URLDecoder.decode(tip);
    long m=Long.parseLong(sm);
    long n=Long.parseLong(sn);
    System.out.println(m+" : "+n+" : "+tip0);
    long c=obj.cmmdc(m,n);
    String message=(new Long(c)).toString();
```

```
30      Response r=null;
31      switch(tip0){
32        case "text/plain":
33          r=Response.ok(getPlainRep(message),"text/plain").build();
34          break;
35        case "text/html":
36          r=Response.ok(getHtmlRep(message),"text/html").build();
37          break;
38      }
39      return r;
40    }

42    public String getPlainRep(String msg) {
43       return msg;
44    }

46    public String getHtmlRep(String msg) {
47      return "<html><head></head><body bgcolor=\"#aaeeaa\"><center>
48         <h1>Cmmdc : "+ msg+"</h1></center></body></html>";
49    }
50  }
```

Este recomandat ca serviciul RESTful să implementeze funcţiile CRUD (Create, Read, Update, Delete) asociindu-le, respectiv cu metodele PUT, GET, POST, DELETE.

Aplicaţii de verificare a unui server REST

- *Advanced REST Client* disponibil in *Chrome Web Store*
- *restclient-ui-*-jar-with-dependencies.jar*

```
java -jar restclient-ui-*-jar-with-dependencies.jar
```

4.2.5 Dezvoltare prin *maven*

Dezvoltarea aplicaţiei server

Dezvoltarea aplicaţiei Web constă din:

1. Generarea aplicaţiei

```
set groupId=resources
set artifactid=hw
mvn archetype:generate
   -DgroupId=%groupId%
   -DartifactId=%artifactId%
   -DarchetypeArtifactId=jersey-quickstart-webapp
   -DarchetypeGroupId=org.glassfish.jersey.archetypes
   -DinteractiveMode=false
   -DarchetypeVersion=2.4.1
```

2. Se adaptează structura de cataloage şi fişiere la

```
hello
|--> src
|    |--> main
|    |    |--> java
|    |    |    |--> HelloWorldResource.java
|    |    |--> resources
|    |    |--> webapp
|    |    |    |--> WEB-INF
|    |    |    |    |    web.xml
|    |    |    |    index.html
|    pom.xml
```

3. Fişierul `pom.xml` se completează cu

```
<build>
    <finalName>HelloWorld</finalName>
    <plugins>
        <plugin>
            <groupId>org.apache.maven.plugins</groupId>
            <artifactId>maven-compiler-plugin</artifactId>
            <version>2.5.1</version>
            <inherited>true</inherited>
            <configuration>
              <source>1.6</source>
              <target>1.6</target>
            </configuration>
        </plugin>
    </plugins>
</build>

<dependencyManagement>
    <dependencies>
        <dependency>
            <groupId>org.glassfish.jersey</groupId>
            <artifactId>jersey-bom</artifactId>
            <version>${jersey.version}</version>
            <type>pom</type>
            <scope>import</scope>
        </dependency>
    </dependencies>
</dependencyManagement>

<dependencies>
    <dependency>
        <groupId>org.glassfish.jersey.containers</groupId>
        <artifactId>jersey-container-servlet-core</artifactId>
        <!-- use the following artifactId if you don't need servlet 2.x compatibility -->
        <!-- artifactId>jersey-container-servlet</artifactId -->
    </dependency>
    <!-- uncomment this to get JSON support
    <dependency>
        <groupId>org.glassfish.jersey.media</groupId>
        <artifactId>jersey-media-moxy</artifactId>
    </dependency>
    -->
</dependencies>
```

```
<properties>
    <jersey.version>2.0</jersey.version>
    <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
</properties>
```

4. Prelucrarea constă din

   (a) `mvn clean package`

   (b) Fişierul `war` care rezultă se desfăşoară în serverul Web.

Dezvoltarea aplicaţiei client

Dezvoltarea aplicaţiei constă din:

1. Generarea aplicaţiei

```
mvn archetype:generate
   -DarchetypeArtifactId=maven-archetype-quickstart
   -DgroupId=hw
   -DartifactId=client
   -DinteractiveMode=false
```

2. Se adaptează structura de cataloage şi fişiere la

```
hello-client
|--> src
|    |--> main
|    |    |--> java
|    |    |    |--> hw
|    |    |    |     |     JerseyClient.java
|    |    |    |     |     Client.java
|    pom.xml
```

3. Fişierul `pom.xml` se completează cu

```
<build>
    <plugins>
        <plugin>
            <groupId>org.apache.maven.plugins</groupId>
            <artifactId>maven-compiler-plugin</artifactId>
            <version>2.5.1</version>
            <inherited>true</inherited>
            <configuration>
              <source>1.7</source>
              <target>1.7</target>
            </configuration>
        </plugin>
    </plugins>
</build>
<dependencies>
   . . .
  <dependency>
```

```
        <groupId>org.glassfish.jersey.core</groupId>
        <artifactId>jersey-client</artifactId>
        <version>${jersey.version}</version>
    </dependency>
    <dependency>
        <groupId>org.glassfish.jersey.connectors</groupId>
        <artifactId>jersey-apache-connector</artifactId>
        <version>${jersey.version}</version>
    </dependency>
  . . .
  </dependencies>
  <properties>
     <jersey.version>2.4.1</jersey.version>
     <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
 </properties>
```

4. Prelucrarea constă din

   (a) `mvn clean compile`

   (b) `mvn exec:java -Dexec.mainClass="hw.JerseyClient"`
   respectiv
   `mvn exec:java -Dexec.mainClass="hw.Client"`

Partea III

MODELUL OSGi

Capitolul 5

Modelul *OSGi*

OSGi - Open Sourse Gateway initiative, 1999, (semnificaţia numelui fiind astăzi depăşită) a dezvoltat un model de cadru de lucru (platformă) privind:

- gestiunea ciclului de viaţă a unei aplicaţii (application life cycle management);
- registru de servicii;
- mediu de execuţie;
- module.

Pe această bază au fost dezvoltate interfeţe de programare (API), servicii, extensii OSGi (OSGi layers).

Cadrul de lucru conţine un model specific de aplicaţie sub formă de componentă sau modul OSGi (bundle for deployment). O asemenea componentă poate pune la dispoziţia altor componente funcţionalităţi, comportându-se ca un serviciu (ofertant de servicii) sau poate executa o acţiune punctuală. O componentă OSGi se prezintă sub forma unei arhive `jar`.

În esenţă, scopul unui cadru de lucru OSGi este oferirea unui mediu pentru crearea şi integrarea uniformă de unităţi (module, componente) de soft.

Un modul este dat de

- O interfaţă de programare (API), pentru rezolvarea unei problemei (opţional);
- O familie de clase realizate în vederea soluţionării unei probleme;
- Mulţimea resurselor necesare claselor.

Un modul în tehnologia OSGi se va înregistra ca un serviciu, într-un *registru de servicii* specific cadrului de lucru.

Crearea unei asemenea serviciu presupune definirea unei interfaţe şi a clasei (claselor) care o implementează, definind două componente OSGi.

O componentă OSGi se poate instala, lansa în execuţie, opri, actualiza şi dezinstala.

Cadrul de lucru conţine un registru de servicii care permite unei componente OSGi să sesizeze existenţa, apariţia sau dispariţia unor servicii.

Programarea unei aplicaţii (serviciu) OSGi se poate face în mod

- *imperativ* prin existenţa unei clase ce implementează interfaţa `org.osgi.framework.BundleActivator`.
- *declarativ* prin utilizarea unor resurse OSGi suplimentare.

5.1 Cadre de lucru OSGi

Există mai multe implementări a modelului OSGi:

- *apache-felix*
- *equinox*
- *Knopflerfish*
- *apache-karaf*

Fiecare cadru de lucru oferă un mediu OSGi.

OSGi prin *apache-felix*

Instalarea constă în dezarhivarea arhivei descărcate din Internet.

Utilizare. Din catalogul unde s-a instalat *apache-felix*, mediul se lansează prin

```
java -jar bin\felix.jar
```

În catalogul în care s-a instalat *apache-felix* se va crea un subcatalog *felix-cache* care este folosit de cadrul de lucru.

Comenzile OSGi sunt

Comanda	Funcţionalitatea
`lb`	Afişează lista modulelor OSGi instalate.
`exit <int>`	Părăseşte şi închide cadrul de lucru.
`install file:`*modulOSGi*`.jar`	Instalează modulul OSGi La instalare unui modul i se atribuie în vederea identificării un număr natural *id.*
`start` *id*	Lansează modulul OSGi *id.*
`start file:` *modulOSGi*`.jar`	Instalează şi lansează modulul OSGi
`stop` *id*	Opreşte modulul OSGi *id.*
`uninstall` *id*	Dezinstalează modulul OSGi *id.*

Interfaţa de lucru, *Apache Felix Gogo,* implementează RFC (*Request for Comments*) 147 publicat de *Internet Engineering Task Force* (IETF).

OSGi prin *Equinox*

Instalare. Dintr-o distribuţie *Eclipse,* se copiază în catalogul propriu fişierul *org.eclipse.osgi_*.jar,* schimbând numele în *equinox.jar.* Alternativ, fişierul menţionat mai sus se poate descărca de la *http://download.eclipse.org/eclipse/equinox.*

Utilizare.

1. *Equinox* a adoptat interfaţa *Apache Felix Gogo.* În vederea lanşrii se va crea structura

```
equinox
|--> configuration
|      |    config.ini
|      org.eclipse.equinox.console_*.jar
|      org.apache.felix.gogo.runtime_*.jar
|      org.apache.felix.gogo.shell_*.jar
|      org.apache.felix.gogo.command_*.jar
|      equinox.jar
```

unde `config.ini` are codul

```
1 osgi.bundles=file\:org.eclipse.equinox.console_*.jar@start,
2    file:\org.apache.felix.gogo.runtime_*.jar@start,
3    file:\org.apache.felix.gogo.shell_*.jar@start,
4    file:\org.apache.felix.gogo.command_*.jar@start
```

2. Consola OSGi iniţială se obţine pe structura

```
equinox
|--> configuration
|      |    config.ini
|      equinox.jar
```

unde `config.ini` are codul

```
1 osgi.console.enable.builtin=true
```

Comenzile OSGi care diferă de cele utilizate de *apache-felix* sunt

Comanda	Funcţionalitatea
`ss`	*short status* – Afişează lista modulelor OSGi instalate.
`exit`	Părăseşte şi închide cadrul de lucru.

În ambele cazuri lansarea se obţine prin

```
java -jar equinox.jar -console
```

apărând prompt-ul `osgi>`.

Cadrul de lucru foloseşte subcatalogul *configuration*.

OSGi prin *knopflerfish*

Instalarea constă în dezarhivarea arhivei descărcate din Internet, reprezentat de un fişier `jar` executabil.

Utilizare. Din catalogul *...\knopflerfish_osgi_*.*.*\osgi* se lansează `java -jar framework.jar`. Toate comenzile de operare sunt lansate dintr-o interfaţa grafică.

5.2 Programare imperativă - Crearea unui modul OSGi

Un modul sau componentă OSGi (bundle) se poate afla în starile `UNINSTALLED, INSTALLED, RESOLVED, STARTING, STOPPING, ACTIVE,` constante definite de interfaţa `org.osgi.framework.Bundle`. Doar cadrul de lucru OSGi poate instanţia o componentă OSGi.

O componentă OSGi este compusă din

1. O clasă ce implementează interfaţa `BundleActivator`.

   Interfaţa `BundleActivator` declară metodele

   - `public void start(BundleContext` *ctx*`)`
     Activităţi executate la lansarea modulului OSGi.

- `public void stop(BundleContext` *ctx*`)`
  Activităţi executate la oprirea modulului OSGi.

Dacă se defineşte un modul interfaţă atunci această clasă lipseşte.

2. Un fişier text *manifest*`.mf` de proprietăţi (nume: valoare) ale modulului OSGi, cu extensia `mf`. Acest fişier trebuie să se termine cu o linie vidă.

   Proprietăţi OSGi

Numele proprietăţii	Semnificaţia
`Bundle-Name`	Numele modului OSGi
`Bundle-Version`	Versiunea modulului OSGi
`Bundle-Activator`	Clasa modulului care implementează interfaţa `BundleActivator`
`Bundle-SymbolicName`	Numele simbolic atribuit modulului OSGi
`Bundle-Description`	Descrierea modulului OSGi
`Manifest-Version`	Versiunea tipului de *manifest*
`Bundle-ManifestVersion`	Versiunea *manifestului* ataşat modulului OSGi
`Bundle-Vendor`	Producătorul modulului OSGi
`Bundle-Classpath`	Cale către resursele suplimentare utilizate
`Import-Package`	Lista pachetelor utilizate de modulul OSGi
`Export-Package`	Lista pachetelor ce pot fi utilizate de alte module OSGi

Structura arhivei `jar` a unei componente OSGi este

```
structura de cataloage corespunzatoare pachetului aplicatiei
| . . .   *.class
META-INF
|  manifest.mf
```

Programarea se bazează pe interfaţa `BundleContext,` care este implementat de fiecare cadru de lucru OSGi în parte.

Metode

- `ServiceRegistration registerService(String` *clazz,* `Object` *service,* `java.util.Dictionary` *properties*`)`

  Înregistrează în cadrul de lucru serviciul definit de obiectul *service,* de tip *clazz* şi cu proprietăile adiacente date de al treilea parametru. Rezultatul este folosit de cadrul de lucru OSGi.

- `ServiceReference[] getServiceReferences(String` *clazz,* `String` *filter*`) throws InvalidSyntaxException`

  Returnează referinţele de tip *clazz.*

- `Object getService(ServiceReference` *reference*`)`

  Returnează obiectul corespunzător referinţei.

5.3 Exemple

Exemplul 5.3.1 *Calculul celui mai mare divizor comun a două numere naturale - ca serviciu OSGi.*

Aplicaţia se compune din 3 module OSGi:

- interfaţă;

  1. Clasa interfeţei *cmmdc.ICmmdc.java*

```
1 package cmmdc;

3 public interface ICmmdc{
4   public long cmmdc(long m,long n);
5 }
```

  2. Fişierul *manifest.mf*

```
1 Bundle-ManifestVersion: 2
2 Bundle-SymbolicName: icmmdc
3 Bundle-Name: Interfata Cmmdc
4 Bundle-Version: 1.0.0
5 Export-Package: cmmdc;version="1.0.0"
```

- serviciu care implementează interfaţa;

  1. Clasa *cmmdc.service.Activator* implementează interfaţa `BundleActivator.` În metoda `start,` se înregistrează o instanţă a clasei ce implementează interfaţa ca serviciu a cadrului OSGi.

```
1 package cmmdc.service;
2 import cmmdc.ICmmdc;
3 import org.osgi.framework.BundleActivator;
4 import org.osgi.framework.BundleContext;

6 public class Activator implements BundleActivator{
7   public void start(BundleContext context){
8     context.registerService(ICmmdc.class.getName(),
9       new CmmdcService(),null);
```

```
10      System.out.println("Registering Cmmdc service.");
11    }

13    public void stop(BundleContext context) {}
14  }
```

2. Clasa *cmmdc.service.CmmdcService* implementează interfaţa *ICmmdc*

```
1  package cmmdc.service;

3  public class CmmdcService implements cmmdc.ICmmdc{
4    public long cmmdc(long m, long n){. . .}
5  }
```

3. Fişierul *manifest.mf*

```
1  Bundle-ManifestVersion: 2
2  Bundle-SymbolicName: cmmdcservice
3  Bundle-Name: Cmmdc Service
4  Bundle-Version: 1.0.0
5  Bundle-Activator: cmmdc.service.Activator
6  Import-Package: org.osgi.framework, cmmdc
```

- client care utilizează interfaţa.

  Prezentăm două variante de modul OSGi client.

1. Referinţa obiectului serviciu se obţine prin intermediul obiectului *context*:`BundleContext`.

   (a) Clasa care implementează interfaţa `BundleActivator` este *cmmdc.client.Activator.java*. În metoda `start,` prin context se obţine o referinţă la serviciul care implementează interfaţa *ICmmdc,* apoi se foloseşte funcţionalitatea obţinută - apelând metoda *cmmdc.*

```
1  package cmmdc.client;
2  import cmmdc.ICmmdc;
3  import java.util.Scanner;
4  import org.osgi.framework.BundleActivator;
5  import org.osgi.framework.BundleContext;
6  import org.osgi.framework.ServiceReference;

8  public class Activator implements BundleActivator{
9    public void start(BundleContext context){
10     try{
11       ServiceReference[] refs=
12         context.getServiceReferences(ICmmdc.class.getName(),
13           null);
14       if(refs!=null){
15         ICmmdc obj=(ICmmdc)context.getService(refs[0]);
```

```
16          Scanner scanner=new Scanner(System.in);
17          System.out.println("Client Cmmdc 1");
18          System.out.println("Primul numar : ");
19          long m=scanner.nextLong();
20          System.out.println("al doilea numar : ");
21          long n=scanner.nextLong();
22          long c=obj.cmmdc(m,n);
23          System.out.println("Cmmdc = "+c);
24        }
25      }
26      catch(Exception e){
27        System.out.println("Client Exception : "+e.getMessage());
28      }
29    }

31    public void stop(BundleContext context) {}
32  }
```

(b) Fişierul *manifest.mf*

```
1  Bundle-ManifestVersion: 2
2  Bundle-SymbolicName: clientcmmdc1
3  Bundle-Name: Cmmdc Client
4  Bundle-Version: 1.0.0
5  Bundle-Activator: cmmdc.client.Activator
6  Import-Package: org.osgi.framework, cmmdc
```

2. Referinţa obiectului serviciu se obţine prin intermediul unui obiect de tip `ServiceTracker,` clasă introdusă pentru simplificarea lucrului cu componente OSGi.

   Constructor

   – `public ServiceTracker(BundleContext` *context,* `String` *clazz,* `ServiceTrackerCustomizer` *customizer*`)`

   Metode

   – `public void open()`

   – `public Object getService()`

   (a) Clasa care implementează interfaţa `BundleActivator` este *cmmdc.client.Activator.java*. În metoda `start,` se crează o instanţa de tip `ServiceTracker` corespunzătoare serviciului generat de interfaţa *ICmmdc*, care este pornită prin metoda `open` şi în final se obţine o referinţa a serviciului, adică o instanţă a clasei care implementează interfaţa *ICmmdc.*

```
1  package cmmdc.client;
2  import cmmdc.ICmmdc;
3  import java.util.Scanner;
4  import org.osgi.framework.BundleActivator;
5  import org.osgi.framework.BundleContext;
```

```
6  import org.osgi.util.tracker.ServiceTracker;

8  public class Activator implements BundleActivator{
9    public void start(BundleContext context){
10     ServiceTracker serviceTracker=new ServiceTracker(context,
11       ICmmdc.class.getName(),null);
12     serviceTracker.open();
13     ICmmdc obj=(ICmmdc)serviceTracker.getService();
14     System.out.println("Client Cmmdc 2");
15     Scanner scanner=new Scanner(System.in);
16     System.out.println("Primul numar : ");
17     long m=scanner.nextLong();
18     System.out.println("al doilea numar : ");
19     long n=scanner.nextLong();
20     long c=obj.cmmdc(m,n);
21     System.out.println("Cmmdc = "+c);
22   }

24   public void stop(BundleContext context) {}
25 }
```

(b) Fişierul *manifest.mf*

```
1 Bundle-ManifestVersion: 2
2 Bundle-SymbolicName: clientcmmdc2
3 Bundle-Name: Cmmdc Client
4 Bundle-Version: 1.0.0
5 Bundle-Activator: cmmdc.client.Activator
6 Import-Package: org.osgi.framework,org.osgi.util.tracker,cmmdc
```

Să presupunem că s-au generat cele module OSGi sub forma

interfaţa	*cmmdc.jar*
serviciul (implementarea interfeţei)	*cmmdcservice.jar*
client	*client1cmmdc.jar*

Execuţia în mediul *apache-felix* constă din

```
start file:d:/. . ./icmmdc.jar
start file:d:/. . ./cmmdcservice.jar
start file:d:/. . ./clientcmmdc1.jar
```

5.4 Dezvoltare OSGi prin *apache-maven*

Codurile claselor Java rămân nemodificate, iar fişierele `manifest.mf` vor fi generate de maven, dar conţinutul lor va trebui specificat în fişierele *pom.xml.*

Exemplificăm cu aplicaţia de calcul a celui mai mare divizor comn, dezvoltată în capitolul *OSGi.*

Într-un catalog, de exemplu - *cmmdc,* se generează trei module `maven`

```
mvn archetype:create -DgroupId=cmmdc -DartifactId=interface
   -DarchetypeArtifactId=maven-archetype-quickstart
```

```
mvn archetype:create -DgroupId=cmmdc.service -DartifactId=impl
   -DarchetypeArtifactId=maven-archetype-quickstart

mvn archetype:create -DgroupId=cmmdc -DartifactId=interface
  -DarchetypeArtifactId=maven-archetype-quickstart
```

În cataloagele generate clasa App.java se înlocuieşte cu clasele aplicaţiei, după cum urmează

Catalog	Clase
interface	ICmmdc.java
impl	CmmdcService.java
	Activator.java
client	Activator.java

iar fişiere `pom.xml` cu cele reproduse mai jos.

Fişierul `pom.xml` din catalogul *interface*

```
<project xmlns="http://maven.apache.org/POM/4.0.0"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
  http://maven.apache.org/xsd/maven-4.0.0.xsd">
  <modelVersion>4.0.0</modelVersion>

  <groupId>cmmdc</groupId>
  <artifactId>icmmdc</artifactId>
  <version>1.0.0</version>
  <packaging>bundle</packaging>

  <name>interface</name>
  <url>http://maven.apache.org</url>

  <properties>
    <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
  </properties>

  <dependencies>
    <dependency>
      <groupId>junit</groupId>
      <artifactId>junit</artifactId>
      <version>3.8.1</version>
      <scope>test</scope>
    </dependency>
    <dependency>
       <groupId>org.osgi</groupId>
       <artifactId>org.osgi.core</artifactId>
       <version>4.2.0</version>
    </dependency>
  </dependencies>
  <build>
    <plugins>
      <plugin>
        <groupId>org.apache.felix</groupId>
        <artifactId>maven-bundle-plugin</artifactId>
        <version>2.0.1</version>
        <extensions>true</extensions>
```

```
39          <configuration>
40            <instructions>
41              <Bundle-SymbolicName>
42                ${project.artifactId}
43              </Bundle-SymbolicName>
44              <Export-Package>
45                cmmdc
46              </Export-Package>
47            </instructions>
48          </configuration>
49        </plugin>
50      </plugins>
51    </build>
52 </project>
```

Fişierul `pom.xml` din catalogul *impl*

```
1  <project xmlns="http://maven.apache.org/POM/4.0.0"
2    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
3    xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
4    http://maven.apache.org/xsd/maven-4.0.0.xsd">
5    <modelVersion>4.0.0</modelVersion>

7    <groupId>cmmdc</groupId>
8    <artifactId>cmmdcservice</artifactId>
9    <version>1.0.0</version>
10   <packaging>bundle</packaging>

12   <name>impl</name>
13   <url>http://maven.apache.org</url>

15   <properties>
16     <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
17     <path>. . ./cmmdc</path>
18   </properties>

20   <dependencies>
21     <dependency>
22       <groupId>junit</groupId>
23       <artifactId>junit</artifactId>
24       <version>3.8.1</version>
25       <scope>test</scope>
26     </dependency>
27     <dependency>
28        <groupId>org.osgi</groupId>
29        <artifactId>org.osgi.core</artifactId>
30        <version>4.2.0</version>
31     </dependency>
32     <dependency>
33       <groupId>icmmdc-1.0.0</groupId>
34       <artifactId>cmmdc.icmmdc</artifactId>
35       <version>1.0.0</version>
36       <scope>system</scope>
37       <systemPath>${path}/interface/target/icmmdc-1.0.0.jar</systemPath>
38     </dependency>
39   </dependencies>
40   <build>
41     <plugins>
42       <plugin>
```

```
43          <groupId>org.apache.felix</groupId>
44          <artifactId>maven-bundle-plugin</artifactId>
45          <version>2.0.1</version>
46          <extensions>true</extensions>
47          <configuration>
48            <instructions>
49              <Bundle-SymbolicName>
50                ${project.artifactId}
51              </Bundle-SymbolicName>
52              <Import-Package>
53                org.osgi.framework,cmmdc,
54              </Import-Package>
55              <Bundle-Activator>cmmdc.service.Activator</Bundle-Activator>
56              <Export-Package/>
57            </instructions>
58          </configuration>
59        </plugin>
60      </plugins>
61    </build>
62  </project>
```

Fişierul `pom.xml` din catalogul *client*

```
1  <project xmlns="http://maven.apache.org/POM/4.0.0"
2    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
3    xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
4    http://maven.apache.org/xsd/maven-4.0.0.xsd">
5    <modelVersion>4.0.0</modelVersion>

7    <groupId>cmmdc</groupId>
8    <artifactId>clientcmmdc</artifactId>
9    <version>1.0.0</version>
10   <packaging>bundle</packaging>

12   <name>client</name>
13   <url>http://maven.apache.org</url>

15   <properties>
16     <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
17     <path>. . ./cmmdc</path>
18   </properties>

20   <dependencies>
21     <dependency>
22       <groupId>junit</groupId>
23       <artifactId>junit</artifactId>
24       <version>3.8.1</version>
25       <scope>test</scope>
26     </dependency>
27      <dependency>
28        <groupId>org.osgi</groupId>
29        <artifactId>org.osgi.core</artifactId>
30        <version>4.2.0</version>
31     </dependency>
32     <dependency>
33       <groupId>icmmdc-1.0.0</groupId>
34       <artifactId>cmmdc.icmmdc</artifactId>
35       <version>1.0.0</version>
36       <scope>system</scope>
```

```
37        <systemPath>${path}/interface/target/icmmdc-1.0.0.jar</systemPath>
38      </dependency>
39    </dependencies>
40    <build>
41      <plugins>
42        <plugin>
43          <groupId>org.apache.felix</groupId>
44          <artifactId>maven-bundle-plugin</artifactId>
45          <version>2.0.1</version>
46          <extensions>true</extensions>
47          <configuration>
48            <instructions>
49              <Bundle-SymbolicName>
50                ${project.artifactId}
51              </Bundle-SymbolicName>
52              <Import-Package>
53                org.osgi.framework,cmmdc,
54              </Import-Package>
55              <Bundle-Activator>cmmdc.client.Activator</Bundle-Activator>
56              <Export-Package/>
57            </instructions>
58          </configuration>
59        </plugin>
60      </plugins>
61    </build>
62  </project>
```

În catalogul aplicaţiei se introduce fişierul *pom.xml*

```
1  <?xml version="1.0" encoding="UTF-8"?>
2  <project>
3    <modelVersion>4.0.0</modelVersion>
4    <artifactId>cmmdc</artifactId>
5    <groupId>cmmdc</groupId>
6    <version>1.0.0</version>
7    <name>Simple cmmdc</name>
8    <packaging>pom</packaging>
9    <modules>
10         <module>interface</module>
11         <module>impl</module>
12         <module>client</module>
13   </modules>
14 </project>
```

şi se lansează comanda maven `mvn clean package`, în urma căreia se crează componentele OSGi ale aplicaţiei.

5.5 Programare declarativă

Modelul declarativ de realizare a unui modul OSGi nu mai necesită realizarea unei clase care să implementeze interfaţa `BundleActivator`. Prezenţa fişierului `manifest.mf` rămâne obligatorie. Există mai multe variante dezvoltate de programare declarativă:

- *Declarative Service*
- *Blueprint*
- *iPOJO*

5.5.1 Programare declarativă prin *Declarative Service*

Modelul de programare este funcţional pe platformele *apache-felix* şi *equinox,* dar resursele suplimentare necesare a fi instalate sunt diferite.

Cadrul *OSGi*	Componenta *OSGi* (*.jar)
felix	org.osgi.compendium-4.*.*.jar
	org.apache.felix.scr-*.jar
equinox	org.eclipse.osgi.services_3.*.*.v*.jar
	org.eclipse.equinox.util_*.jar
	org.eclipse.equinox.ds_*.jar

Cu excepţia interfeţelor, programatorul trebuie să editeze un fişier de configurare `OSGI-INF\component.xml,` care se include în arhiva `jar` a componentei OSGi. Acest fişierul de configurare este menţionat în fişierul `manifest.mf` prin linia

```
Service-Component:  OSGI-INF/component.xml
```

Astfel arhiva modulului OSGi are structura

```
structura de cataloage corespunzatoare pachetului aplicatiei
| . . .   *.class
META-INF
|    manifest.mf
OSGI-INF
|    component.xml
```

Un serviciu programat prin modelul declarativ se poate apela de către un client programat prin modelul imperativ, dar se poate programa şi un modul OSGi client prin model declarativ.

Exemplul 5.5.1 *Calculul celui mai mare divizor comun a două numere naturale.*

Aplicaţia este alcătuită din cele trei componente:

- interfaţa

1.
```
package cmmdc;

public interface ICmmdc{
  public long cmmdc(long m,long n);
}
```

2. *manifest.mf*

```
Bundle-ManifestVersion: 2
Bundle-SymbolicName: icmmdc
Bundle-Name: Interfata Cmmdc
Bundle-Vendor: unitbv
Bundle-Version: 1.0.0
Export-Package: cmmdc;version="1.0.0"
```

- clasa serviciului OSGi, care implementează interfaţa

1.
```
package cmmdc.service;

public class CmmdcService implements cmmdc.ICmmdc{
  public long cmmdc(long m, long n){. . .}
}
```

2. *manifest.mf*

```
Bundle-ManifestVersion: 2
Bundle-SymbolicName: cmmdcservice
Bundle-Name: Cmmdc Service
Bundle-Version: 1.0.0
Import-Package: cmmdc
Service-Component: OSGI-INF/component.xml
```

3. *component.xml*

```
<?xml version="1.0" encoding="UTF-8"?>
<scr:component xmlns:scr="http://www.osgi.org/xmlns/scr/v1.1.0"
        name="DS Service Sample">
  <implementation class="cmmdc.service.CmmdcService"/>
  <service>
    <provide interface="cmmdc.ICmmdc"/>
  </service>
</scr:component>
```

Client declarativ

1.
```
package cmmdc.client;
import java.util.Scanner;
import cmmdc.ICmmdc;

public class CmmdcClient {
  private ICmmdc service;

  public void bindMyService(ICmmdc a) {
    System.out.println("Service was set");
    service = a;
    Scanner scanner=new Scanner(System.in);
    System.out.println("m=");
    long m=scanner.nextLong();
    System.out.println("n=");
    long n=scanner.nextLong();
    System.out.println("Cmmdc = "+service.cmmdc(m,n));
  }

  public void unbindMyService(ICmmdc a) {
    if(service==a)
      service = null;
    System.out.println("Service was unset");
  }
}
```

2. *manifest.mf*

```
Bundle-ManifestVersion: 2
Bundle-SymbolicName: clientds
Service-Component: OSGI-INF/component.xml
Private-Package: cmmdc.client
Bundle-Name: Declarative Service Cmmdc Client
Bundle-Description: DS Cmmdc Client
Bundle-Version: 1.0.0
Import-Package: cmmdc
```

3. *component.xml*

```
<?xml version="1.0" encoding="UTF-8"?>
<scr:component xmlns:scr="http://www.osgi.org/xmlns/scr/v1.1.0"
        name="DS Client">
   <implementation class="cmmdc.client.CmmdcClient"/>
   <reference interface="cmmdc.ICmmdc"
       name="ICmmdc" cardinality="1..1" policy="static"
       bind="bindMyService"
       unbind="unbindMyService"/>
</scr:component>
```

Sunt declarate metodele care

- consumă serviciul (`bind=...`);
- disponibilizează serviciul injectat (`unbind=...`).

5.5.2 Programare declarativă prin *Blueprint*

Modelul de programare este funcţional pe platformele *apache-felix* şi *equinox.* Implementarea de referinţă este dată în proiectul *apache-aries.*

Cadrul *OSGi*	Componenta *OSGi* (*.jar)
felix & equinox	org.apache.felix.configadmin-1.6.0.jarr
	pax-logging-api-1.6.4.jarr
	pax-logging-service-1.6.4.jar
	aries-blueprint-1.0.0-incubating-SNAPSHOT.jar

Toate parţile unei aplicaţii (interfeţe, implementările lor, clienţii) trebuie să facă parte din aceaşi componentă OSGi. Un client este reprezentat de două clase:

- Înregistratorul (*listener*);

  Mediul OSGi asigură serviciul, adică un obiect care implementează interfată aplicaţiei. Trebuie programate două metode, una în care se utilizează serviciul (uzual se apelează un clientul propriu-zis) şi a doua în care se disponibilizeaza serviciul. Cele două metode sunt precizate în fişierul de configurare `OSGI-INF\blueprint\config.xml.`

- Clientul propriu-zis.

Serviciile înregistrate pot fi apelate şi de un client programat imperativ.

În afara fişierului `manifest.mf` mai este necesar fişierul de configurare `OSGI-INF\blueprint\config.xml.`

Astfel arhiva modulului OSGi are structura

```
structura de cataloage corespunzatoare pachetului aplicatiei
| . . .   *.class
META-INF
|    manifest.mf
OSGI-INF
|--> blueprint
|      |     config.xml
```

Exemplul 5.5.2 *Calculul celui mai mare divizor comun a două numere naturale.*

Aplicaţia va conţine:

- Clasele *cmmdc.ICmmdc* şi *cmmdc.service.CmmdcService* amintite anterior.
- Clasa *listener cmmdc.client.RegistrationListener*

```
1  package cmmdc.client;
2  import java.util.Map;
3  import java.util.Scanner;
4  import cmmdc.ICmmdc;

6  public class RegistrationListener {
7    public void register(ICmmdc obj, Map properties) {
8      VisualCmmdcClient myApp=new VisualCmmdcClient(obj);
9    }

11   public void unregister(ICmmdc obj, Map properties) {
12     System.out.println("Cmmmdc service unregistered");
13   }
14 }
```

  Clasa *VisualCmmdcClient* este un client cu interfaţăa grafică. (Se va utiliza `setDefaultCloseOperation(javax.swing.WindowConstants.HIDE_ON_CLOSE).`)

- *manifest.mf*

```
1 Bundle-ManifestVersion: 2
2 Bundle-SymbolicName: blueprint
3 Bundle-Name: Blueprint Cmmdc Client
4 Bundle-Description: Blueprint Cmmdc App
5 Bundle-Version: 1.0.0
6 Import-Package: org.osgi.framework,javax.swing,
7   org.osgi.service.blueprint;version="[1.0.0,2.0.0)"
8 Export-Package: cmmdc,blueprint;
9   uses:="org.osgi.framework";version="1.0.0.SNAPSHOT"
```

- *OSGI-INF\blueprint\config.xml*

```
1  <?xml version="1.0" encoding="UTF-8"?>
2  <blueprint xmlns="http://www.osgi.org/xmlns/blueprint/v1.0.0">
3     <bean id="cmmdc"
4         class="cmmdc.service.CmmdcService" scope="prototype"/>

6     <service id="serviceClient" ref="cmmdc">
7        <interfaces>
8          <value>cmmdc.ICmmdc</value>
9        </interfaces>
10       <registration-listener
11               registration-method="register"
12               unregistration-method="unregister">
13           <bean class="cmmdc.client.RegistrationListener"/>
14       </registration-listener>
15    </service>
16 </blueprint>
```

5.5.3 Programare declarativă prin *apache-iPOJO*

Acest model este funcţional pe platforma OSGi *apache felix.* În etapa de dezvoltare conţinutul fişierului `manifest.mf` este extins cu date dintr-un alt fişier *metadata.xml* - aspectul declarativ al modelului de programare. Datele acestui fişier definesc componenta OSGi care se înregistrează pe platforma OSGi dată de *apache-felix.*

Completarea fişierului `manifest.mf` se face automat, cu instrumente specifice *apache-ant* sau *apache-maven.*

Exemplul 5.5.3 *Calculul celui mai mare divizor comun a două numere naturale.*

Aplicaţia este alcătuită din cele trei componente:

- interfaţa

```
1 package cmmdc;

3 public interface ICmmdc{
4   public long cmmdc(long m,long n);
5 }
```

- clasa serviciului OSGi, care implementează interfaţa

```
1 package cmmdc.service;

3 public class CmmdcService implements cmmdc.ICmmdc{
4   public long cmmdc(long m, long n){. . .}
5 }
```

metadata.xml

```
1 <?xml version="1.0" encoding="UTF-8"?>
2 <ipojo
3     xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
4     xsi:schemaLocation="org.apache.felix.ipojo
5     http://felix.apache.org/ipojo/schemas/CURRENT/core.xsd"
6     xmlns="org.apache.felix.ipojo">

8   <component classname="cmmdc.service.CmmdcService" name="CmmdcProvider">
9     <provides />
10  </component>

12  <instance component="CmmdcProvider" name="CmmdcService" />
13 </ipojo>
```

- client OSGi

```
1  package cmmdc.client;
2  import cmmdc.ICmmdc;

4  public class CmmdcClient implements Runnable {

6      /**
7       *  Intarziere intre doua apelari automate
8       */
9      private static final int DELAY = 10000;


12     /**
13      *  Reprezentant al serviciului injectat de iPOJO
14      */
15     private ICmmdc m_cmmdc;

17     /**
18      *  Indicator pentru depistarea starsitului
19      */
20     private boolean m_end;

22     /**
23      * Argumente pentru metodele invocate
24      * injectate de containerul OSGi
25      */
26     private String sm;
27     private String sn;

29     /**
30      * Metoda run (java.lang.Runnable)
31      */
32     public void run() {
33         while (!m_end) {
34             try {
35                 invokeCmmdcServices();
36                 Thread.sleep(DELAY);
37             } catch (InterruptedException ie) {
38            /*  se verifica conditia de sfarsit */
39             }
40         }
41     }

43     /**
44      * Invocarea serviciului
45      */
46     public void invokeCmmdcServices() {
47         int i=0;
48         long m=Long.parseLong(sm);
49         long n=Long.parseLong(sn);
50        System.out.println(m_cmmdc.cmmdc(m,n));
51        m_end=true;
52     }

54     /**
55      * Start
56      */
57     public void starting() {
58         Thread thread = new Thread(this);
```

```
59          m_end = false;
60          thread.start();
61      }

63      /**
64       * Stop
65       */
66      public void stopping() {
67          m_end = true;
68      }
69  }
```

metadata.xml

```
1  <?xml version="1.0" encoding="UTF-8"?>
2  <ipojo
3      xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
4      xsi:schemaLocation="org.apache.felix.ipojo
5      http://felix.apache.org/ipojo/schemas/CURRENT/core.xsd"
6      xmlns="org.apache.felix.ipojo">

8    <component classname="cmmdc.client.CmmdcClient">
9      <requires field="m_cmmdc" />
10     <callback transition="validate" method="starting" />
11     <callback transition="invalidate" method="stopping" />
12     <properties>
13       <property field="sm" name="sm" />
14       <property field="sn" name="sn" />
15     </properties>
16   </component>

18   <instance component="cmmdc.client.CmmdcClient">
19     <property name="sm" value="56" />
20     <property name="sn" value="42" />
21   </instance>
22 </ipojo>
```

Execuţia constă din instalarea şi lansarea componentelor

```
start file:. . ./org.apache.felix.ipojo-1.8.4.jar
start file:. . ./org.apache.felix.shell-1.4.3.jar
start file:. . ./org.apache.felix.ipojo.arch-1.6.0.jar

start file:. . ./icmmdc.jar
start file:. . ./cmmdcservice.jar
start file:. . ./clientcmmdc.jar
```

Dezvoltare cu *apache-ant*

Dezvoltarea se bazează pe `org.apache.felix.ipojo.ant-*.jar`. Codul `build.xml` în cazul dezvoltării serviciului este

```
1  <project name="cmmdcservice" basedir="." default="main">
2    <property file="../build.properties"/>

4    <path id="myclasspath">
```

```
5     <!--<pathelement path="${OSGi}.jar"/>-->
6     <pathelement path="../interface/dist/icmmdc.jar"/>
7   </path>

9   <taskdef name="ipojo"
10          classname="org.apache.felix.ipojo.task.IPojoTask"
11          classpath="../lib/org.apache.felix.ipojo.ant-*.jar"/>

13  <target name="init">
14    <delete dir="${dist.dir}"/>
15    <delete dir="${build.dir}"/>
16    <mkdir dir="${build.dir}"/>
17    <mkdir dir="${dist.dir}" />
18  </target>

20  <target name="compile" depends="init">
21    <javac classpathref="myclasspath"
22           includeantruntime="false"
23           srcdir="src"
24           destdir="${build.dir}"/>
25  </target>

27  <target name="generate.jar" depends="compile">
28    <jar destfile="${dist.dir}/${ant.project.name}.jar"
29      manifest="${manifest.name}.mf"
30      basedir="${build.dir}"/>
31  </target>

33  <target name="main" depends="generate.jar">
34    <ipojo
35        input="dist/cmmdcservice.jar"
36        metadata = "metadata.xml"/>
37  </target>
38 </project>
```

Fişierele `manifest.mf` necesare exemplului sunt:

- pentru interfaţa

  manifest.mf

```
1 Bundle-ManifestVersion: 2
2 Bundle-SymbolicName: icmmdc
3 Bundle-Name: Interfata Cmmdc
4 Bundle-Vendor: unitbv
5 Bundle-Version: 1.0.0
6 Export-Package: cmmdc;version="1.0.0"
```

- pentru implementarea serviciului OSGi

  manifest.mf

```
1 Bundle-ManifestVersion: 2
2 Bundle-SymbolicName: cmmdcservice
3 Bundle-Name: Cmmdc Service
4 Bundle-Version: 1.0.0
5 Import-Package: cmmdc
```

- pentru client

  manifest.mf

```
Bundle-ManifestVersion: 2
Bundle-SymbolicName: clientcmmdc
Bundle-Name: CmmdcClient
Bundle-Description: Simple Cmmdc Client
Bundle-Vendor: Apache Felix
Bundle-Version: 1.0.0
Import-Package: cmmdc
```

Dezvoltare cu *apache-maven*

Reproducem fişierele *pom.xml* utilizate pentru dezvoltarea celor trei componente. Datele care se trec în într-un fişiere `manifest.mf` sunt definite în fişierul `pom.xml`.

- interfaţa

```
<project>
  <modelVersion>4.0.0</modelVersion>
  <packaging>bundle</packaging>
  <groupId>cmmdc</groupId>
  <artifactId>icmmdc</artifactId>
  <version>1.0.0</version>
  <name>Cmmdc interface</name>

  <build>
    <plugins>
      <plugin>
        <groupId>org.apache.felix</groupId>
        <artifactId>maven-bundle-plugin</artifactId>
        <version>2.0.1</version>
        <extensions>true</extensions>
        <configuration>
          <instructions>
            <Bundle-SymbolicName>
              ${pom.artifactId}
            </Bundle-SymbolicName>
            <Export-Package>
              cmmdc
            </Export-Package>
          </instructions>
        </configuration>
      </plugin>
    </plugins>
  </build>
</project>
```

- serviciu

```
<project>
  <modelVersion>4.0.0</modelVersion>
```

```
3   <packaging>bundle</packaging>
4   <groupId>cmmdc.impl</groupId>
5   <artifactId>cmmdc.service</artifactId>
6   <version>1.0.0</version>
7   <name>Cmmdc Service Provider</name>

9   <dependencies>
10    <dependency>
11      <groupId>cmmdc</groupId>
12      <artifactId>icmmdc</artifactId>
13      <version>1.0.0</version>
14    </dependency>
15  </dependencies>

17  <build>
18    <plugins>
19      <plugin>
20        <groupId>org.apache.felix</groupId>
21        <artifactId>maven-bundle-plugin</artifactId>
22        <version>2.0.1</version>
23        <extensions>true</extensions>
24        <configuration>
25          <instructions>
26            <Bundle-SymbolicName>
27              ${pom.artifactId}
28            </Bundle-SymbolicName>
29            <Private-Package>
30              cmmdc.service
31            </Private-Package>
32          </instructions>
33        </configuration>
34      </plugin>
35      <plugin>
36        <groupId>org.apache.felix</groupId>
37        <artifactId>maven-ipojo-plugin</artifactId>
38        <version>1.6.0</version>
39        <executions>
40          <execution>
41            <goals>
42              <goal>ipojo-bundle</goal>
43            </goals>
44          </execution>
45        </executions>
46      </plugin>
47    </plugins>
48  </build>
49</project>
```

- client

```
1<project>
2   <modelVersion>4.0.0</modelVersion>
3   <packaging>bundle</packaging>
4   <groupId>cmmdc.impl</groupId>
5   <artifactId>cmmdc.service</artifactId>
6   <version>1.0.0</version>
7   <name>Cmmdc Service Provider</name>
```

```
 9   <dependencies>
10     <dependency>
11       <groupId>cmmdc</groupId>
12       <artifactId>icmmdc</artifactId>
13       <version>1.0.0</version>
14     </dependency>
15   </dependencies>

17   <build>
18     <plugins>
19       <plugin>
20         <groupId>org.apache.felix</groupId>
21         <artifactId>maven-bundle-plugin</artifactId>
22         <version>2.0.1</version>
23         <extensions>true</extensions>
24         <configuration>
25           <instructions>
26             <Bundle-SymbolicName>
27               ${pom.artifactId}
28             </Bundle-SymbolicName>
29             <Private-Package>
30               cmmdc.service
31             </Private-Package>
32           </instructions>
33         </configuration>
34       </plugin>
35       <plugin>
36         <groupId>org.apache.felix</groupId>
37         <artifactId>maven-ipojo-plugin</artifactId>
38         <version>1.6.0</version>
39         <executions>
40           <execution>
41             <goals>
42               <goal>ipojo-bundle</goal>
43             </goals>
44           </execution>
45         </executions>
46       </plugin>
47     </plugins>
48   </build>
49 </project>
```

5.6 Serviciul OSGi de jurnalizare

Interfaţa de programare (API) OSGi conţine pentru jurnalizare o serie de interfeţe a căror implementare oferă servicii de jurnalizare.

În cazul mediului *apache-felix* serviciile de jurnalizare trebuie instalate, resursa fiind

org.apache.felix.log-.jar*

Interfaţa `org.osgi.service.log.LogService`

Clasa ca implementează această interfaţă asigură jurnalizarea. Mesajele sunt reţinute de mediul OSGi.

Sunt declarate nivelele `LogService.LOG_ERROR, LogService.LOG_WARNING, LogService.LOG_INFO, LogService.LOG_DEBUG.`

Metode

- `void log(int` *level,* `String` *message*`)`

Interfaţa `org.osgi.service.log.LogReaderService`

Are ca scop preluarea mesajelor de jurnalizare. Uzual aceste mesaje sunt prelucrate de un *ascultător,* un obiect care implementează interfaţa `LogListener`

Metode

- `void addLogListener(LogListener` *listener*`)`

Interfaţa `org.osgi.service.log.LogListener`

Metode

- `void logged(LogEntry` *entry*`)`

Instalarea tuturor serviciilor se obţine cu o componenta OSGi având activatorul

```
1  import org.osgi.framework.BundleActivator;
2  import org.osgi.framework.BundleContext;
3  import org.osgi.framework.ServiceReference;
4  import org.osgi.service.log.LogService;
5  import org.osgi.service.log.LogReaderService;
6  import org.osgi.service.log.LogListener;
7  import org.osgi.service.log.LogEntry;

9  public class Activator implements BundleActivator{

11   public void start(BundleContext context) throws Exception {
12     ServiceReference ref =
13       context.getServiceReference(LogService.class.getName());
14     LogService logService=null;
15     if (ref != null){
16       logService = (LogService)context.getService(ref);
17     }
18     ref = context.getServiceReference(LogReaderService.class.getName());
19     if (ref != null){
20       LogReaderService reader = (LogReaderService) context.getService(ref);
21       reader.addLogListener(new LogWriter());
22     }
23     logService.log(LogService.LOG_INFO, "Pornirea serviciului de jurnalizare");
24   }
```

```
26    public void stop(BundleContext context) throws Exception{}

28    class LogWriter implements LogListener{
29      public void logged(LogEntry entry){
30        System.out.println(entry.getMessage());
31      }
32    }
33  }
```

împreună cu fişierul manifest

```
1  Bundle-ManifestVersion: 2
2  Bundle-SymbolicName: LogService
3  Bundle-Name: LogService
4  Bundle-Version: 1.0.0
5  Bundle-Activator: Activator
6  Import-Package: org.osgi.framework,org.osgi.service.log
```

5.7 *Apache-karaf*

Apache-karaf este un mediu OSGi, care integrează o serie de funcţionalităţi OSGi (blueprint, http, etc.). Folosim denumirea simplificată *Karaf,* pentru mediul de lucru.

Instalarea produsului constă în dezarhivarea fişierului descărcat.

Lansarea mediului este

```
set JAVA_HOME=. . .
set KARAF_HOME=. . .
del %KARAF_HOME%\lock
del %KARAF_HOME%\instances\*
rmdir %KARAF_HOME%\instances
%KARAF_HOME%\bin\karaf.bat clean
```

În catalogul %KARAF_HOME% se generează catalogul *instances* iar în fereastra DOS va apare prompt-ul `karaf@root>`.

Oprirea se obţine apăsând tastele CTRL+D.

Karaf posedă o consolă DOS dar şi o consolă Web. Consola Web trebuie instalată

`feature:install webconsole`

Consola Web se apelează dintr-un navigator prin

http://localhost:8181/system/console

Pentru a schimba portul se crează în prealabil fişierul `etc\org.ops4j.pax.web.cfg` cu conţinutul

`org.osgi.service.http.port`=8080

Instalarea componentelor OSGi se poate face

- copiindu-le în catalogul
  `%KARAF_HOME%\deploy`
- în mod obişnuit, prin comanda `install file:...`

  În acest caz fişierul `MANIFEST.mf` trebuie să conţină atributele

```
Bundle-ManifestVersion: 2
Bundle-SymbolicName: numeComponentaOSGi
```

Comenzi OSGi uzuale: `start` n, `stop` n, `install file:...`, `uninstall` n, `list`, `help`.

Karaf permite executarea componentelor OSGi programate imperativ şi descriptiv prin *blueprint,* dar fără necesitatea instalării unor resurse suplimentare.

Capitolul 6

OSGi distribuit

Integrarea unei aplicaţii Web într-o platformă *OSGi* necesită o abordare specifică. Integrată într-o platformă *OSGi,* aplicaţia Web nu mai este desfăşurată în serverul Web, dar apelurile se vor adresa în continuare serverului Web. În consecinţă, este nevoie de o punte între platforma *OSGi* şi serverul Web, care eventual să asigure funcţionalităţi suplimentare.

6.1 Medii OSGi pentru aplicaţii distribuite

Apache-Karaf

Mediul OSGi *apache-karaf* utilizează serverul Web incorporat *Jetty.* Se va instala suportul pentru protocolul `http`

```
feature:install http
```

Glassfish

Mediul *OSGi* are la bază platforma *apache-felix.* Glassfish oferă posibilităţi *OSGi* pentru serviciile

- *HttpService*
- *TransactionService*
- *JDBC Data Source Service*
- *JMS Resource Service*

Comenzile OSGi se apelează prin
`asadmin osgi` *comanda_OSGi*

Equinox Bridge Servlet

Equinox Bridge Servlet[1] este distribuit printr-o arhivă `war`, destinată a fi desfăşurată într-un server Web precum *apache-tomcat* sau *jetty.* În felul acesta *Equinox Bridge Servlet* integrează platforma *OSGi equinox* într-un server Web container de servlet. *Equinox Bridge Servlet* lansează platforma *equinox.* Consola *OSGi* se obţine apăsând tasta `Enter` în fereastra DOS ataşată serverului Web.

Funcţionarea corectă a servlet-ului *Equinox Bridge Servlet* se verifică prin apelul `http://localhost:8080/bridge/sp_test`. Bineînţeles, în acest caz s-a presupus că serverul Web în care s-a desfăşurat *Equinox Bridge Servlet* este calculatorul local.

Platforma *equinox* instalată în serverul Web se poate instala, dezinstala, reinstala, porni şi opri prin apelurile `http://`*host:port*`/bridge/`*numeApel,* unde *numeApel* are valorile, respectiv `sp_deploy, sp_undeploy, sp_redeploy, sp_start, sp_stop.`

6.2 Servlet ca modul OSGi

Prezentăm integrarea unei aplicaţii servlet într-o componentă *OSGi* bazată pe interfaţa *HttpService.* Cu foarte puţine diferenţe componenta *OSGi* se va putea utiliza pe platforme *OSGi* diferite *apache-karaf, glassfish, bridge.* Clasa servlet-ului rămâne nemodificată.

Integrarea servlet-ului prin HttpService

Interfaţa `org.osgi.service.http.HttpService` declară:
Metode

- `void registerResources(String` *alias,* `String` *name,* `HttpContext` *context*`) throws NamespaceException`
- `void registerServlet(String` *alias,* `Servlet` *servlet,* `Dictionary` *init-params,* `HttpContext` *context*`)ServletException, NamespaceException`
- `void unregister(String` *alias*`)`

Structura componentei OSGi corespunzătoare unui servlet este Structura componentei OSGi este

[1] www.eclipse.org/equinox/server/http_in_container.php

```
|--> META-INF
|    |    MANIFEST.MF
|    ClasaServlet.class
|    Activator.class
|    fisier.html
```

Rămâne la latitudinea programatorului să includă sau nu pagina `html`.

Compilarea necesită accesul la unul din pachetele care implementează interfaţa *HttpService:*

- `org.eclipse.osgi.services_3.*.*.v*.jar.`
- `org.apache.felix.http.bundle-*.jar`

Prezentăm două moduri de programare în câte un exemplu.

Exemplul 6.2.1 *Integrarea servlet-ului HelloServlet.*

Aplicaţia servlet este alcătuită din clasa `HelloServlet.java` şi o pagină `html` de apelare *helloname.html.*

Clasa *Activator* are codul

```
1  import org.osgi.framework.BundleActivator;
2  import org.osgi.framework.BundleContext;
3  import org.osgi.framework.ServiceReference;
4  import org.osgi.service.http.HttpService;

6  public class Activator implements BundleActivator{

8    public void start(BundleContext context) throws Exception{
9      ServiceReference sRef =
10        context.getServiceReference(HttpService.class.getName());
11     if (sRef != null){
12       HttpService service = (HttpService) context.getService(sRef);
13       service.registerServlet("/hello", new HelloServlet(), null, null);
14       service.registerResources("/helloname.html", "/index.html", null);
15     }
16   }

18   public void stop(BundleContext context) throws Exception{}
19 }
```

Fişierul `manifest.mf` al componentei *OSGi* este

```
1  Manifest-Version: 1.0
2  Bundle-ManifestVersion: 2
3  Bundle-Name: HelloServlet
4  Bundle-SymbolicName: HelloServlet
5  Bundle-Version: 1.0.0
6  Bundle-Activator: Activator
7  Import-Package: javax.servlet,
8     javax.servlet.http,
9     org.osgi.framework;version="1.3.0",
10    org.osgi.service.http;version="1.2.0"
```

Apelarea servlet-ului din pagina `html` este dependentă de platforma *OSGi* utilizată:

Cadrul *OSGi*	Apel (aplicaţie, servlet)
karaf	*http://host:8080/helloname.html* * /hello*
glassfish	*http://host:8080/osgi/helloname.html* * /osgi/hello*
Equinox Bridge Servlet	*http://host:port/bridge/helloname.html* * /bridge/hello*

bridge este numele de apel definit de *Equinox Bridge Servlet* în fişierul `web.xml`.

A două variantă de programare se bazează pe utilizarea clasei `ServiceTracker` pentru înregistrarea servlet-ului.

Exemplul 6.2.2 *Integrarea servlet-ului CmmdcServlet.*

Aplicaţia servlet este alcătuită din clasa `CmmdcServlet.java` şi o pagină `html` de apelare *cmmdc.html.*

Clasa *Activator* are codul

```
1  import org.osgi.framework.BundleActivator;
2  import org.osgi.framework.BundleContext;
3  import org.osgi.framework.ServiceReference;
4  import org.osgi.service.http.HttpService;
5  import org.osgi.util.tracker.ServiceTracker;

7  public class Activator implements BundleActivator{
8    private ServiceTracker httpServiceTracker;

10   public void start(BundleContext context) throws Exception{
11     httpServiceTracker = new HttpServiceTracker(context);
12     httpServiceTracker.open();
13   }

15   public void stop(BundleContext context) throws Exception {
16     httpServiceTracker.close();
17     httpServiceTracker = null;
18   }

20   private class HttpServiceTracker extends ServiceTracker{

22     public HttpServiceTracker(BundleContext context){
23       super(context, HttpService.class.getName(), null);
24     }

26     public Object addingService(ServiceReference reference){
27       HttpService httpService = (HttpService)context.getService(reference);
28       try {
29         httpService.registerResources("/cmmdc.html","/cmmdc.html",null);
30         httpService.registerServlet("/cmmdc",new CmmdcServlet(),null,null);
```

```
31        }
32        catch (Exception e){
33          e.printStackTrace();
34        }
35        return httpService;
36      }

38      public void removedService(ServiceReference reference, Object service){
39        HttpService httpService = (HttpService) service;
40        httpService.unregister("/cmmdc.html");
41        httpService.unregister("/cmmdc");
42        super.removedService(reference, service);
43      }
44    }
45  }
```

Serverul Web va recunoaşte apelul către fişierul *cmmdc.html* şi către servlet prin numele de apel *cmmdc.html* şi respectiv *cmmdc* - liniile de cod 29-30.

Fişierul `manifest.mf` al componentei *OSGi* este

```
1  Manifest-Version: 1.0
2  Bundle-ManifestVersion: 2
3  Bundle-Name: CmmdcServlet
4  Bundle-SymbolicName: CmmdcServlet
5  Bundle-Version: 1.0.0
6  Bundle-Activator: Activator
7  Bundle-Localization: plugin
8  Import-Package: javax.servlet, javax.servlet.http,
9   org.osgi.framework;version="1.3.0",
10  org.osgi.service.http;version="1.2.0",
11  org.osgi.util.tracker;version="1.3.1"
```

Variantă descriptivă în Bridge

Clasa care implementează interfaţa `BundleActivator` este înlocuită cu un fişier de configurare `plugin.xml`. Structura componentei OSGi este

```
|--> META-INF
|    |    MANIFEST.MF
|    ClasaServlet.class
|    fisier.html
|    plugin.xml
```

Pentru exemplul anterior fişierul `manifest.mf` este

```
1  Manifest-Version: 1.0
2  Bundle-ManifestVersion: 2
3  Bundle-Name: CmmdcServlet PlugIn
4  Bundle-SymbolicName: CmmdcServlet PlugIn;singleton:=true
5  Bundle-Version: 1.0.0
6  Bundle-Localization: plugin
7  Import-Package: javax.servlet, javax.servlet.http
8  Require-Bundle: org.eclipse.equinox.http.registry
```

iar conţinutul fişierului `plugin.xml` este

```
<?xml version="1.0" encoding="UTF-8"?>
<?eclipse version="3.0"?>
<plugin>
  <extension-point id="servlets"
    name="HttpService servlets"
    schema="schema/servlets.exsd"/>
  <extension-point id="resources"
    name="HttpService resources"
    schema="schema/resources.exsd"/>
  <extension-point id="httpcontexts"
    name="HttpService httpcontexts"
    schema="schema/httpcontexts.exsd"/>
  <extension
    id="myServlet"
    point="org.eclipse.equinox.http.registry.servlets">
    <servlet
        alias="/plugin/cmmdc"
        class="CmmdcServlet">
    </servlet>
  </extension>
  <extension
    id="myResource"
    point="org.eclipse.equinox.http.registry.resources">
    <resource
      alias="/plugin/cmmdc.html"
      base-name="/cmmdc.html"
    />
  </extension>
</plugin>
```

Se va proceda la fel ca în varianta anterioară cu observaţia că adresele de apelare, definite de atributul `alias`, vor fi

```
http://host:port/bridge/plugin/cmmdc.html
http://host:port/bridge/plugin/cmmdc
```

Partea IV

JAVA MANAGEMENT EXTENSIONS

Capitolul 7

Java Management Extensions

Java Management Extensions (JMX) face posibilă ca un obiect Java să permită gestionarea metodelor şi a anumitor câmpuri de către alte obiecte. Obiectele Java care îşi expun astfel resursele se numesc MBean - uri şi formează temelia cadrului de lucru JMX.

Există mai multe tipuri de MBean-uri:

- Standard MBean;
- Dynamic MBean;
- Open MBean;
- Model MBean;
- MXBean;

Un obiect care gestionează resursele unui MBean se numeşte *agent* sau *server MBean.* Agentul dispune de mijloace care interacţionează cu un MBean, permiţându-i:

- accesul la valorile unui câmp şi la modificarea lor;
- invocarea metodelor.

În general, un agent poate fi definit ca

- autorul unei acţiuni;
- factor care provoacă acţiuni;
- reprezentant al unei instituţii însărcinat cu îndeplinirea unor acţiuni.

Terminologia *server MBean* se justifică prin faptul că poate fi invocat de un program client. În această ipostază, serverul MBean are rolul unui container de MBean-uri şi de gestionare şi execuţie a apelurilor clienţilor.

Structura unei aplicaţii JMX conţine două nivele:

- componentele MBean;
- agentul (serverul MBean).

7.1 Standard MBean

7.1.1 Crearea unui Standard MBean

O componentă MBean este alcătuită dintr-o interfaţă şi o clasă care implementează interfaţa satisfăcând următoarele restricţii:

1. interfaţa are numele clasei care o implementează având în plus sufixul `MBean;`
2. Interfaţa şi clasa care o implementează aparţin aceluiaşi pachet;
3. constructorii şi metodele expuse trebuie să fie publice.

În continuare câmpurile şi metodele destinate expunerii se vor denumi atribute, respectiv operaţii. Fiecărui atribut xxx i se ataşează cel puţin una din metodele

```
public void setXxx(tip xxx){
  this.xxx=xxx;
}
```

şi / sau

```
public tip getXxx(){
  return xxx;
}
```

Un atribut se precizează doar prin aceste metode, fără definirea / declararea câmpului corespunzător. Câmpul se defineşte în clasa ce implementează interfaţa MBean-ului.

Astfel, un MBean este caracterizat de

- atribute

  care pot fi consultate (citite), modificate (scrise) sau cu ambele opţiuni.

- operaţii

- notificări

  cu evidenţa modificărilor suferite de atribute.

Exemplul 7.1.1

Construim un MBean cu

- atributele
  - label
    ce poate fi numai citit;
  - cursEuro
    care poate fi consultat şi modificat;
- operaţiile
  - `public String sayHello()`
    afişează un mesaj;
  - `public long cmmdc(long m, long n);`
    de calcul a celui mai mare divizor comun a două numere naturale.

Interfaţa *IntroMBean* este

```
1  package basic;
2
3  public interface IntroMBean {
4      // Operatii
5      public String sayHello();
6      public long cmmdc(long m, long n);
7
8      // Atribute
9      // read-only
10     public String getLabel();
11     // read-write
12     public double getCursEuro();
13     public void setCursEuro(double cursEuro);
14 }
```

fiind implementat de clasa *Intro*

```
1  package basic;
2
3  public class Intro implements IntroMBean {
4
5    //Atribute
6    private final String label = "Fac. Matematica si Informatica";
7    private double cursEuro = 4.50;
8
9    public String getLabel() {
10     return label;
11   }
12
13   public double getCursEuro() {
14     return cursEuro;
15   }
16
17   public synchronized void setCursEuro(double cursEuro) {
18     this.cursEuro = cursEuro;
19   }
20
21   //Operatii
22   public String sayHello() {
23     String message="Hello World !";
24     System.out.println(message);
25     return message;
26   }
27
28   public long cmmdc(long m,long n){. . .}
29 }
```

În acest caz nu s-a implementat posibilitatea notificărilor atributului *cursEuro.*

7.1.2 Crearea unui MBeanServer

Un agent sau MBean server implementează interfaţa `MBeanServer.` Un asemenea obiect se obţine printr-una din metodele statice

- `static MBeanServer ManagementFactory.getPlatformMBeanServer()`

  Utilitarul *jconsole* permite conectarea la serverul MBean.

- `static MBeanServer MBeanServerFactory.createMBeanServer()`

În agent se înregistrează componente MBean. Un obiect MBean este caracterizat de un *nume,* un obiect de tip `ObjectName.` Înregistrarea şi / sau crearea unei componente MBean necesită definirea în prealabil a acestui *nume.* Structura unei *nume* este

domeniu : numeAtribut=valAtribut ,numeAtribut=valAtribut . . .

unde

- *domeniu*

  este un nume simbolic (String). Dacă domeniul este stringul vid atunci se consideră valoarea implicită *DefaultDomain.*

- atribute uzuale:

  – `type=`*numele MBean-ului*
  – `index=`*număr de identificare a MBean-ului*

  Cel puţin un atribut este obligatoriu.

Clasa `ObjectName`

Constructori

- `ObjectName(String` *nume*`)`

  Parametrul *nume* are structura descrisă mai sus.

- `ObjectName(String` *domeniu*`, Hashtable<String,String>` *tabel*`)`

- `ObjectName(String` *domeniu*`, String` *numeAtribut*`, String` *valAtribut*`)`

Metode

- `static ObjectName getInstance(String` *nume*`)`

Înregistrarea şi utilizarea MBean-ului face apel la metodele interfeţei `MBeanServer`.

ObjectInstance

Un obiect de tip `ObjectInstance` este folosit pentru reprezentarea ansamblului alcătuit de un obiect `ObjectName` asociat unui MBean şi numele clasei corespunzătoare.

Interfaţa `MBeanServer`

Metode

- `ObjectInstance registerMBean(Object` *obj*`, ObjectName` *nume*`)`

  Inregistrează pe platformă, instanţa *obj* a unui MBean având numele *nume*.

- `void unregister(ObjectName` *nume*`)`
- `ObjectInstance createMBean(String` *numeClasă*`, ObjectName` *nume*`)`

  Crează şi înregistrează un MBean de clasă *numeClasă* şi de nume *nume.*
- `ObjectInstance createMBean(String` *numeClasă*`, ObjectName` *nume*`, Object[]` *param*`, String[]` *sign*`)`

  În plus, şirul *param* conţine parametri constructorului, de tip, respectiv *sign.*
- `String getDefaultDomain()`
- `Object invoke(ObjectName` *mbeanName*`, String` *operationName*`, Object[]` *param*`, String[]` *paramTip*`)`

  Se invocă operaţia *operationName* a MBean-ului *mbeanName.* Parametri necesari operaţiei împreună cu tipurile corespunzătoare sunt daţi în variabilele *param* şi respectiv *paramTip.*
- `Object getAttribute(ObjectName` *mbeanName*`,String` *atribut*`)`

  Returnează valoarea atributului *atribut* a mbean-ului *mbeanName.*
- `void setAttribute(ObjectName` *mbeanName*`,Attribute` *atribut*`)`

  Fixează valoarea atributului *atribut* a MBean-ului `mbeanName`. Reţinem doar constructorul clasei `Attribute` prin

  `Attribute(String` *nume*`, Object` *valoare*`)`
- `MBeanInfo getMBeanInfo(ObjectName` *mbeanName*`)`

  Returnează un obiect de tip `MBeanInfo` util inspectării resurselor unui MBean.

Există mai multe şabloane de programare a înregistrării unei componente MBean.

Exemplul 7.1.2 *Se crează două componente MBean de tip Intro care vor fi inspectate prin intermediul utilitarului jconsole*[1] *- distribuţia* ***`jdk`*** *.*

Utilitarul jconsole permite apelarea operaţiilor, modificarea atributelor şi semnalează notificările.

[1] În acest caz, este nevoie ca variabilele de tip clasă acoperitoare `Double, Long` să fie înlocuite prin tipuri predefinite.

```
1  package basic;
2  import java.lang.management.ManagementFactory;
3  import javax.management.ObjectName;
4  import javax.management.MBeanServer;

6  public class Main{
7    public static void main(String[] args){
8      String domeniu="";
9      if(args.length>0) domeniu=args[0];
10     try{
11       // Serverul platformei
12       MBeanServer mbs = ManagementFactory.getPlatformMBeanServer();

14       //Varianta 1
15       // Construirea ObjectName corespunzator MBean-ului
16       ObjectName mbeanObjectName =
17         new ObjectName(domeniu+":type=Intro,index=1");

19       // Crearea MBean-ului
20       Intro mbean = new Intro();

22       // Inregistrarea MBean-ului
23       mbs.registerMBean(mbean, mbeanObjectName);

25       //Varianta 2
26       mbeanObjectName=new ObjectName(domeniu+":type=Intro,index=2");
27       mbs.createMBean("basic.Intro",mbeanObjectName);

29       // Asteptare nedefinita
30       System.out.println("Waiting forever...");
31       while(true);
32     }
33     catch(Exception e){
34       System.err.println("Exception : "+e.getMessage());
35     }
36   }
37 }
```

Executarea aplicaţiei revine la

1. Într-o fereastră DOS se lansează agentul *Main* prin
   `java -Dcom.sun.manager.jmxremote basic.Main`
2. Într-o altă fereastră DOS se lansează
   `jconsole`
   pornind utilitarul *jconsole.*
3. În fereastra de dialog *jconsole:Connect to Agent* se dă clic pe `Connect.`
4. În panoul *Tree* găsim domeniul dat. Prin clic pe domeniu apar MBean-urile de indice 1 şi 2.
5. Prin clic pe unul din aceste MBean-uri, în panoul central avem acces la atributele şi la operaţiile lor.

Rezultatul unei operaţii apare într-o fereastră de informare (Fig. 7.1).

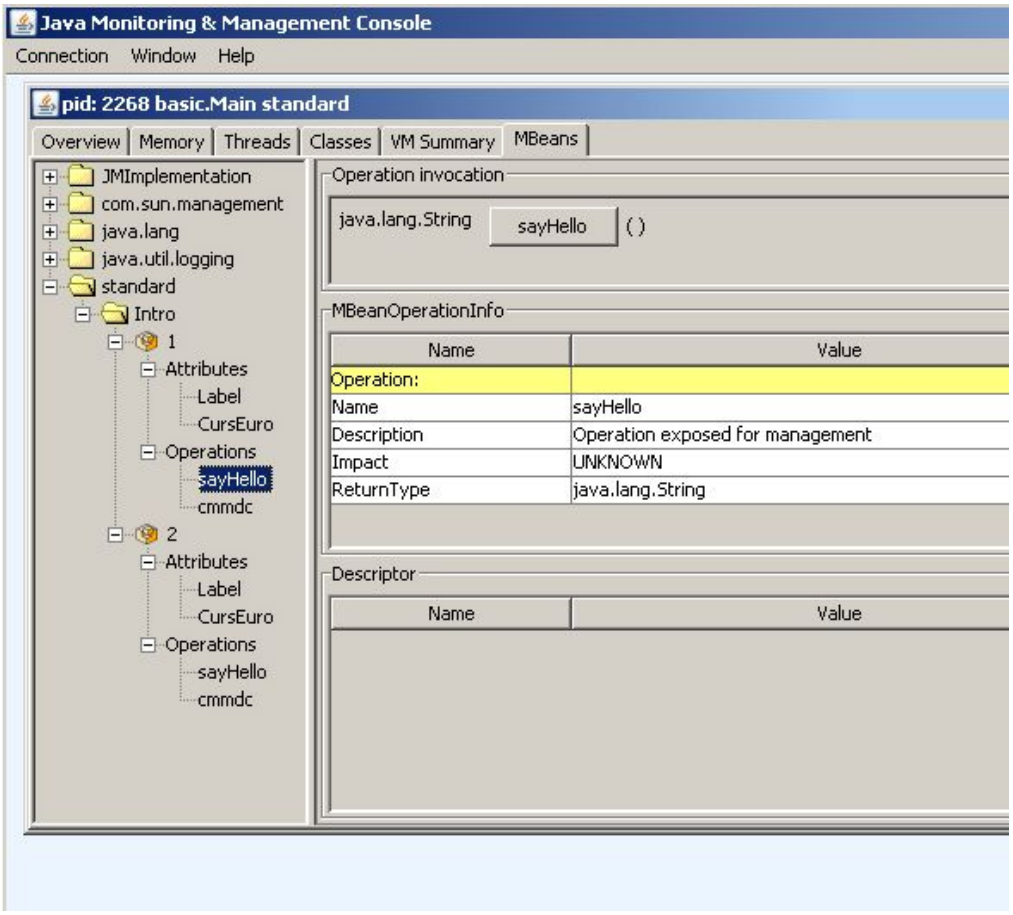

Figure 7.1: Rezultatele furnizate de *jcluster*.

7.1.3 Notificări

Pentru implementarea reţinerii de către un agent MBean a notificărilor, clasa ce implementează interfaţa MBean-ului trebuie să extindă clasa `NotificationBroadcasterSupport`.

Modificările unui atribut se reţin într-un obiect de tip `Notification` şi este transmis prin metoda

`sendNotification(Notification` *notification*`)`

a clasei `NotificationBroadcasterSupport`.

Suplimentar se defineşte metoda

```
public MBeanNotificationInfo[] getNotificationInfo()
```

în care se precizează

- tipul notificării - (constantă definită de clasa `AttributeChangeNotification`);
- clasa în care s-a generat notificarea

  `String` *name* `= AttributeChangeNotification.class.getName();`
- o descriere a modificării.

Astfel, introducerea notificărilor presupune familiarizarea cu clasele

- **NotificationBroadcasterSupport**

  Constructori

  - `NotificationBroadcasterSupport()`

  Metode

  - `void sendNotification(Notification` *notificare*`)`
- **Notification**

  Constructori

  - `Notification(String` *type*`, Object` *source*`, long` *sequenceNumber*`)`
  - `Notification(String` *type*`, Object` *source*`, long` *sequenceNumber*`, long` *timeStamp*`)`
  - `Notification(String` *type*`, Object` *source*`, long` *sequenceNumber*`, long` *timeStamp*`, String` *mesaj*`)`
  - `Notification(String` *type*`, Object` *source*`, long` *sequenceNumber*`, String` *mesaj*`)`

  Metode

  - `public void setUserData(Object` *userData*`)`
  - `public Object getUserData()`

- **AttributeChangeNotification extends Notification**

  Constructori

  – `AttributeChangeNotification(Object` *source*`, long` *sequenceNumber*`, long` *timeStamp*`, String` *msg*`, String` *attributeName*`, String` *attributeType*`, Object` *oldValue*`, Object` *newValuE*`)`

  Câmpuri

  – `static String ATTRIBUTE_CHANGE`
  Semnalează schimbarea atributului

Exemplul 7.1.3 *Extinderea MBean-ului Intro cu notificarea modificărilor atributelor în jconsole.*

Implementarea notificărilor în clasa *Intro* presupune modificarea metodei *setCursEuro.* Totodată se adaugă metoda *getNotificationInfo,* ce furnizează informaţii referitoare la modificarea apărută. Codul clasei care implementează interfaţa devine

```
1  package agentn;
2  import javax.management.Notification;
3  import javax.management.AttributeChangeNotification;
4  import javax.management.NotificationBroadcasterSupport;
5  import javax.management.MBeanNotificationInfo;

7  public class IntroN extends NotificationBroadcasterSupport
8       implements IntroNMBean{
9    private long sequenceNumber=1;

11   //Atribute
12   private final String label = "Fac. Matematica si Informatica";
13   private double cursEuro = 4.50;

15   public String getLabel() {. . .}

17   public double getCursEuro() {. . .}

19   public synchronized void setCursEuro(double cursEuro) {
20     double oldCursEuro=this.cursEuro;
21     this.cursEuro = cursEuro;
22     //System.out.println("Curs de schimb euro : " + euro+" ron.");
23     Notification n=new AttributeChangeNotification(
24       this,
25       sequenceNumber++,
26       System.currentTimeMillis(),
27       "Schimbarea cursului Euro",
28       "cursEuro",
29       "double",
30       oldCursEuro,
31       cursEuro);
```

```
32      sendNotification(n);
33    }

35    public MBeanNotificationInfo[] getNotificationInfo() {
36      String[] types = new String[] {
37        AttributeChangeNotification.ATTRIBUTE_CHANGE
38      };
39      String name = AttributeChangeNotification.class.getName();
40      String description = "An attribute of this MBean has changed";
41      MBeanNotificationInfo info =
42        new MBeanNotificationInfo(types, name, description);
43      return new MBeanNotificationInfo[] {info};
44    }

46    // Operatii
47    public String sayHello() {. . .}

49    public long cmmdc(long m,long n){. . .}
50  }
```

În *jconsole* pentru notificare, în prealabil este nevoie de subscriere.

7.1.4 Agent MBean

În exemplelele anterioare resursele unui MBean au fost utilizate prin *jconsole.* Valorificarea resurselor unui MBean *Intro,* dintr-un MBeanServer (agent) se programează prin

Exemplul 7.1.4

```
1  package agent;
2  import java.io.IOException;
3  import javax.management.ObjectName;
4  import javax.management.MBeanServer;
5  import javax.management.MBeanServerFactory;
6  import javax.management.Attribute;
7  import java.util.Scanner;

9  public class Agent{
10   public static void main(String[] args) {
11     try {
12       // Crearea Agentului - MBeanServer
13       MBeanServer mbs = MBeanServerFactory.createMBeanServer();

15       // Crearea unui MBean
16       String domain = mbs.getDefaultDomain();
17       String className="agent.Intro";
18       String sObjectName=domain+":type="+className;
19       ObjectName mbeanObjectName = new ObjectName(sObjectName);
20       mbs.createMBean(className,mbeanObjectName);

22       // Utilizarea MBean-ului
23       // Apelarea operatiilor
24       String operatia="sayHello";
```

```
25          mbs.invoke(mbeanObjectName,operatia,null,null);

27          operatia="cmmdc";
28          System.out.println("Cmmdc al numerelor:");
29          Scanner scanner=new Scanner(System.in);
30          System.out.println("Primul numar:");
31          long m=scanner.nextLong();
32          System.out.println("Al doilea numar:");
33          long n=scanner.nextLong();
34          Object[] param={m,n};
35          String[] sign={"long","long"};
36          Long r=(Long)mbs.invoke(mbeanObjectName,operatia,param,sign);
37          System.out.println("cmmdc="+r.toString());

39          // Utilizarea Atributelor
40          String label=(String)mbs.getAttribute(mbeanObjectName,"Label");
41          System.out.println("Valoarea atributului label : "+label);

43          System.out.println("Introduceti cursul euro");
44          double cursEuro=scanner.nextDouble();
45          Attribute curs=new Attribute("CursEuro",cursEuro);
46          mbs.setAttribute(mbeanObjectName,curs);
47          Double euro=(Double)mbs.getAttribute(mbeanObjectName,"CursEuro");
48          System.out.println("Valoarea atributului cursEuro : "+euro);
49      }
50      catch (Exception e){
51          System.out.println(e.getMessage());
52      }
53    }
54 }
```

7.1.5 Invocarea la distanţă

Din punct de vedere al programării distribuite, cazul interesant este cel în care clasa ce implementează MBean-ul şi clientul (agentul MBean) care o utilizează se află pe calculatoare diferite.

În acest caz:

- Este nevoie de o clasa *server* (agent) al cărui rol este
  - Instanţierea unui MBeanServer
  - Instanţiarea unui server de conexiune, obiect de tip `MBeanServerConnection`. Bazat pe tehnologia RMI sau RMI-IIOP, acest obiect gestionează comunicaţia dintre un client şi serverul MBeanServer. Serverul de conexiune şi serverul MBeanServer aparţin aceleiaşi clase.

    În cazul utilizării tehnologiei RMI *rmiregistry* şi serverul MBeanServer trebuie să ruleze pe acelaşi calculator.

    Utilizând tehnologia RMI-IIOP, *orbd* şi serverul MBeanServer pot rula pe calculatoare distincte.

– Lansarea în execuţie a serverului de conexiune.

- Clientul dispune de interfaţa MBean-ului.

Interfaţa `MBeanServerConnection` este implementată de clasa `MBeanServer` Şablonul de programare pentru instanţierea obiectului de tip `MBeanServerConnection` şi lansarea sa în execuţie poate fi:

```
String surl="service:jmx:rmi:///jndi/rmi://host:port/numeServer"
// String surl="service:jmx:iiop:///jndi/iiop://host:port/numeServer";

// Crearea unui server-conector
JMXServiceURL url = new JMXServiceURL(surl);
JMXConnectorServer cs =
  JMXConnectorServerFactory.newJMXConnectorServer(url,null,MBeanServer);
//
// Pornirea server-conectorului
cs.start();
System.out.println("Press Enter to finish !");
System.in.read();
cs.stop();
```

Serverul de conexiune are un nume, *numeServer* care trebuie cunoscut de către client.

Lansarea în execuţie a *serverului* este precedată de pornirea registrului *rmiregistry,* respectiv *orbd.*

Exemplul 7.1.5

```
1  package server;
2  import javax.management.MBeanServer;
3  import javax.management.MBeanServerFactory;
4  import javax.management.remote.JMXServiceURL;
5  import javax.management.remote.JMXConnectorServer;
6  import javax.management.remote.JMXConnectorServerFactory;

8  public class MBServer{
9    public static void main(String[] args) {
10     String host="localhost";
11     String port="1099";
12     //String port="1050";
13     if(args.length==0){
14       System.out.println("The name of the server is required");
15       System.exit(0);
16     }
17     if(args.length>=2)
18       host=args[1];
19     if(args.length>=3)
20       port=args[2];
21     try {
22       // Crearea MBeanServer
23       MBeanServer mbs = MBeanServerFactory.createMBeanServer();
```

```
25        // Crearea unui server-conector
26        String surl="service:jmx:rmi:///jndi/rmi://"+
27          host+":"+port+"/"+args[0];
28        // String surl="service:jmx:iiop:///jndi/iiop://"+
29        //    host+":"+port+"/"+args[0];
30        JMXServiceURL url=new JMXServiceURL(surl);
31        JMXConnectorServer cs=
32          JMXConnectorServerFactory.newJMXConnectorServer(url,null,mbs);

34        // Pornirea server-conectorului
35        cs.start();
36        System.out.println("Press Enter to finish !");
37        System.in.read();
38        cs.stop();
39      }
40      catch (Exception e){
41         System.out.println(e.getMessage());
42         e.printStackTrace();
43      }
44    }
45  }
```

Primul argument care trebuie furnizat programului anterior (*args[0]*) este numele serverului.

Clientul, la rândul lui, este nevoit să creeze un *conector* către *server* (agent), prin intermediul căruia obţine un obiect ce implementează interfaţa `MBeanServerConnection`. Prin acest obiect, clientul va putea crea MBean-uri - în agent - şi le va putea utiliza resursele.

Şablonul de programare pentru crearea obiectului `MBeanServerConnection` poate fi

```
JMXServiceURL url =
  new JMXServiceURL("service:jmx:rmi:///jndi/rmi://host:port/numeServer");
JMXConnector jmxc = JMXConnectorFactory.connect(url,null);
// Obtinerea obiectului de tip MBeanServerConnection
MBeanServerConnection cs = jmxc.getMBeanServerConnection();
```

Resursele unui MBean se poate invoca prin

- metoda `invoke(`*mbeanObjectName,operationName,param,sign*`)` a interfeţei `MBeanServerConnection.`
- crearea unui reprezentant local al MBean-ului prin metoda statică

$$\begin{array}{c} \textit{mbeanClass proxy} = \\ \textit{(mbeanClass)}\texttt{MBeanServerInvocationHandler.newProxyInstance(} \\ \textit{mbeanServerConnection,mbeanObjectName,mbeanClass.class,true);} \end{array}$$

În final, MBean-ul se şterge din serverul de conexiune, prin `cs.unregisterMBean(mbeanObjectName)`.

Exemplul 7.1.6 *Client ce utilizează un MBean Intro, definit în Exemplul 7.1.1*

```
1  package client;
2  import java.util.Scanner;
3  import javax.management.MBeanServerConnection;
4  import javax.management.ObjectName;
5  import javax.management.Attribute;
6  import javax.management.MBeanServerInvocationHandler;
7  import javax.management.remote.JMXServiceURL;
8  import javax.management.remote.JMXConnector;
9  import javax.management.remote.JMXConnectorFactory;

11 public class Client {
12   public static void main(String[] args) {
13     String host="localhost";
14     String port="1099";
15     if(args.length<=1){
16       System.out.println("The server and domain names are required");
17       System.exit(0);
18     }
19     String serverName=args[0];
20     String domain=args[1];
21     if(args.length>=3)  host=args[2];
22     if(args.length>=4)  port=args[4];
23     Scanner scanner=new Scanner(System.in);
24     try {
25       //Crearea unui conector si a obiectului de tip MBeanServercsection
26       String surl="service:jmx:rmi:///jndi/rmi://"+
27         host+":"+port+"/"+args[0];
28       // String surl="service:jmx:iiop:///jndi/iiop://"+
29       //    host+":"+port+"/"+args[0];
30       JMXServiceURL url = new JMXServiceURL(surl);
31       JMXConnector jmxc = JMXConnectorFactory.connect(url,null);
32       MBeanServerConnection cs = jmxc.getMBeanServerConnection();

34       // Domeniile agentului sunt
35       System.out.println("Domains:");
36       String domains[] = cs.getDomains();
37       for (int i = 0; i < domains.length; i++) {
38         System.out.println("\tDomain[" + i + "] = " + domains[i]);
39       }

41       // iar domeniul implicit
42       System.out.println("DefaultDomain : " +cs.getDefaultDomain());
43       System.out.println("Domain : " +domain);

45       // Crearea unui MBean Intro
46       String className="basic.Intro";
47       String sObjectName=domain+":type="+className;
48       ObjectName mbeanObjectName = new ObjectName(sObjectName);
49       cs.createMBean(className,mbeanObjectName, null, null);

51       double cursEuro;
52       long m,n;

54       // Utilizarea MBean-ului
55       // Varianta 1 de invocare prin proxy
```

```
56        System.out.println("Varianta de invocare prin proxi");
57        IntroMBean proxy=
58          (IntroMBean)MBeanServerInvocationHandler.newProxyInstance(
59                  cs,
60                  mbeanObjectName,
61                  client.IntroMBean.class,
62                  true);

64        //Utilizarea operatiilor
65        // operatia "sayHello"
66        proxy.sayHello();

68        // operatia cmmdc
69        System.out.println("Cmmdc al numerelor:");
70        System.out.println("Primul numar:");
71        m=scanner.nextLong();
72        System.out.println("Al doilea numar:");
73        n=scanner.nextLong();
74        System.out.println("Cmmdc="+proxy.cmmdc(m,n));

76        // Utilizarea atributelor
77        System.out.println("Numele: "+proxy.getLabel());
78        System.out.println("Introduceti cursul euro");
79        cursEuro=scanner.nextDouble();
80        proxy.setCursEuro(cursEuro);
81        System.out.println("Euro : "+proxy.getCursEuro());

83        // Varianta 2 de invocare prin conexiune
84        System.out.println("Varianta de invocare prin conexiune");
85        // Apelarea operatiilor
86        String operatia="sayHello";
87        cs.invoke(mbeanObjectName,operatia,null,null);
88        operatia="cmmdc";
89        System.out.println("Cmmdc al numerelor:");
90        System.out.println("Primul numar:");
91        m=scanner.nextLong();
92        System.out.println("Al doilea numar:");
93        n=scanner.nextLong();
94        Object[] param={m,n};
95        String[] sign={"long", "long"};
96        Long r=(Long)cs.invoke(mbeanObjectName,operatia,param,sign);
97        System.out.println("Cmmdc="+r.toString());

99        // Utilizarea Atributelor
100       String label=(String)cs.getAttribute(mbeanObjectName,"Label");
101       System.out.println("Valoarea atributului label : "+label);

103       System.out.println("Introduceti cursul euro");
104       cursEuro=scanner.nextDouble();
105       Attribute curs=new Attribute("CursEuro", cursEuro);
106       cs.setAttribute(mbeanObjectName,curs);
107       Double newEuro=(Double)cs.getAttribute(mbeanObjectName,"CursEuro");
108       System.out.println("Valoarea atributului euro : "+newEuro);
109       cs.unregisterMBean(mbeanObjectName);
110     }
111     catch(Exception e) {
112       System.out.println(e.getMessage());
113     }
114   }
```

```
115 }
```

Inspectarea şi valorificarea resurselor unei componente MBean se poate face şi prin intermediul metodei

```
1   private static void getMBeanResources(MBeanInfo info){
2     System.out.println("CLASA: \t" + info.getClassName());
3     System.out.println("DESCR: \t" + info.getDescription());
4     System.out.println("ATTRIBUTE");
5     MBeanAttributeInfo[] attrInfo = info.getAttributes();
6     if (attrInfo.length > 0) {
7       for (int i = 0; i < attrInfo.length; i++) {
8         System.out.println("\tNUME: \t" + attrInfo[i].getName());
9         System.out.println("\tDESC: \t" + attrInfo[i].getDescription());
10        System.out.println("\tTIP: \t" + attrInfo[i].getType() +
11         " READ: "+ attrInfo[i].isReadable() +
12         " WRITE: "+ attrInfo[i].isWritable());
13      }
14    }
15    else
16      System.out.println("\tFara atribute !");
17    System.out.println("CONSTRUCTORI");
18    MBeanConstructorInfo[] constrInfo = info.getConstructors();
19    for (int i=0; i<constrInfo.length; i++) {
20      System.out.println("\tNUME: \t" + constrInfo[i].getName());
21      System.out.println("\tDESCR: \t" + constrInfo[i].getDescription());
22      System.out.println("\tPARAM: \t" +
23        constrInfo[i].getSignature().length +" parametri");
24    }
25    System.out.println("OPERATII");
26    MBeanOperationInfo[] opInfo = info.getOperations();
27    if (opInfo.length > 0) {
28      for (int i = 0; i < opInfo.length; i++) {
29        System.out.println("\tNUME: \t" + opInfo[i].getName());
30        System.out.println("\tDESCR: \t" + opInfo[i].getDescription());
31        System.out.println("\tPARAM: \t" +
32          opInfo[i].getSignature().length +" parametri");
33      }
34    }
35    else
36      System.out.println("\tFara operatii");
37    System.out.println("NOTIFICARI");
38    MBeanNotificationInfo[] notifInfo = info.getNotifications();
39    if (notifInfo.length > 0) {
40      for (int i = 0; i < notifInfo.length; i++) {
41        System.out.println("\tNUME: " + notifInfo[i].getName());
42        System.out.println("\tDESCR: " + notifInfo[i].getDescription());
43        String notifTypes[] = notifInfo[i].getNotifTypes();
44        for (int j = 0; j < notifTypes.length; j++) {
45          System.out.println("\tTIP: " + notifTypes[j]);
46        }
47      }
48    }
49    else
50      System.out.println("\tFara notificari");
51  }
```

Această metodă poate fi inserată în oricare din programele agent sau client.

Notificarea la distanţă

Notificarea la distanţă presupune utilizarea unui MBean posedând această facilitate. Pe partea de client trebuie implementat interfaţa `NotificationListener` care declară metoda

`public void handleNotification(Notification` *notification*`, Object` *handback*`)`

Ataşarea şi disponibilizarea clasei ce implementează interfaţa `Notification Listener` se obţin prin metodele interfeţei `MBeanServerConnection`:

- `void addNotificationListener(ObjectName` *name*`, NotificationListener` *listener*`, NotificationFilter` *filter*`, Object` *handback* `)throws InstanceNotFoundException, IOException`
- `void removeNotificationListener(ObjectName` *name*`, ObjectName` *listener* `)throws InstanceNotFoundException, ListenerNotFoundException, IOException`

Exemplul 7.1.7 *Folosind exemplul 7.1.6 - dar cu MBean-ul creat pentru 7.1.3 se crează un client cu notificare, care sesisează modificarea valoarii atributului cursEuro a MBean-ului IntroN.*

Implementarea interfeţei `NotificationListener` este

```
1  package client;
2  import javax.management.Notification;
3  import javax.management.NotificationListener;
4  import javax.management.AttributeChangeNotification;

6  public class ClientListener implements NotificationListener {
7      public void handleNotification(Notification notification,
8          Object handback) {
9        System.out.println("\nReceived notification: " + notification);
10       AttributeChangeNotification myNotif=
11         (AttributeChangeNotification)notification;
12       System.out.println("Curs initial : " +
13         myNotif.getOldValue().toString());
14       System.out.println("Curs curent  : " +
15         myNotif.getNewValue().toString());
16     }
17 }
```

Codul clasei client este

```
1  package client;
2  import javax.management.ObjectName;
3  import javax.management.MBeanInfo;
4  import javax.management.MBeanAttributeInfo;
5  import javax.management.MBeanConstructorInfo;
```

```
6  import javax.management.MBeanOperationInfo;
7  import javax.management.MBeanNotificationInfo;
8  import javax.management.MBeanServerConnection;
9  import javax.management.remote.JMXServiceURL;
10 import javax.management.remote.JMXConnector;
11 import javax.management.remote.JMXConnectorFactory;

13 public class ClientNotif {
14   public static void main(String[] args) {
15     String host="localhost";
16     String port="1099";
17     if(args.length<=1){
18       System.out.println("The server and domain names are required");
19       System.exit(0);
20     }
21     String serverName=args[0];
22     String domain=args[1];
23     if(args.length>=3) host=args[2];
24     if(args.length>=4) port=args[3];
25     ClientNotif obj=new ClientNotif();
26     try {
27       // Crearea unui conector si a obiectului
28       // de tip MBeanServerConnection
29       String surl="service:jmx:rmi:///jndi/rmi://" +
30         host+":"+port+"/"+args[0];
31       JMXServiceURL url = new JMXServiceURL(surl);
32       JMXConnector jmxc = JMXConnectorFactory.connect(url,null);
33       MBeanServerConnection cs = jmxc.getMBeanServerConnection();

35       // Crearea obiectului OBjectName atasat MBean-ului IntroN
36       String className="basicn.IntroN";
37       String sObjectName=domain+":type="+className;
38       ObjectName mbeanObjectName = new ObjectName(sObjectName);

40       MBeanInfo info=cs.getMBeanInfo(mbeanObjectName);
41       getMBeanResources(info);

43       // Utilizarea notificarii
44       // Crearea unui ascultator
45       ClientListener listener = new ClientListener();
46       // Activarea notificatorului
47       cs.addNotificationListener(mbeanObjectName,listener,null,obj);

49       Thread.sleep(500);
50       // Disponibilizarea ascultatorului de notificari
51       System.out.println("Press Enter to finish !");
52       try{
53         System.in.read();
54       }
55       catch(java.io.IOException e){}
56       cs.removeNotificationListener(mbeanObjectName, listener);
57       // Disponibilizarea obiectului MBeanObjectName
58       cs.unregisterMBean(mbeanObjectName);
59     }
60     catch(Exception e) {
61       System.out.println(e.getMessage());
62       e.printStackTrace();
63     }
64   }
```

```
66    private static void getMBeanResources(MBeanInfo info){. . .}

68  }
```

Clasa *Client* care crează MBean-ul trebuie lansată înaintea clasei *ClientNotif.* Aceste două clase pot rula pe calculatoare distincte.

Exemplul 7.1.8 *O aplicaţie servlet întreţine un cont. Acţiunile ce pot fi întreprinse sunt: depunerea unei sume, extragerea unei sume în limita soldului şi consultarea contului. Contul este implementat ca un MBean standard. Se cere urmărirea la distanţa a modificărilor suferite de cont.*

Interfaţa contului (a MBean-ului) este

```
1  public interface ContMBean {
2    // Atribute
3    // read-write
4    public double getCont();
5    public void setCont(double cont);
6  }
```

implementat prin

```
1  import javax.management.Notification;
2  import javax.management.AttributeChangeNotification;
3  import javax.management.NotificationBroadcasterSupport;
4  import javax.management.MBeanNotificationInfo;

6  public class Cont extends NotificationBroadcasterSupport
7       implements ContMBean{
8    private long sequenceNumber=1;
9    private double cont;

11   public synchronized double getCont() {
12     return cont;
13   }

15   public synchronized void setCont(double cont) {
16     double oldCont=this.cont;
17     this.cont=cont;
18     Notification n=new AttributeChangeNotification(
19       this,
20       sequenceNumber++,
21       System.currentTimeMillis(),
22       "Schimbarea Cont",
23       "cont",
24       "double",
25       oldCont,
26       cont);
27     sendNotification(n);
28   }

30   public MBeanNotificationInfo[] getNotificationInfo() {
31     String[] types = new String[] {
32       AttributeChangeNotification.ATTRIBUTE_CHANGE
```

```
33      };
34      String name = AttributeChangeNotification.class.getName();
35      String description = "An attribute of this MBean has changed";
36      MBeanNotificationInfo info =
37        new MBeanNotificationInfo(types, name, description);
38      return new MBeanNotificationInfo[] {info};
39    }
40  }
```

Codul servlet-ului este

```
1  import java.io.IOException;
2  import javax.servlet.ServletException;
3  import javax.servlet.http.HttpServlet;
4  import javax.servlet.http.HttpServletRequest;
5  import javax.servlet.http.HttpServletResponse;
6  import javax.servlet.ServletOutputStream;
7  import javax.servlet.ServletConfig;
8  import javax.servlet.annotation.WebServlet;
9  import javax.servlet.annotation.WebInitParam;
10 import javax.management.ObjectName;
11 import javax.management.Attribute;
12 import javax.management.MBeanServerConnection;
13 import javax.management.remote.JMXServiceURL;
14 import javax.management.remote.JMXConnector;
15 import javax.management.remote.JMXConnectorFactory;

17 @WebServlet(urlPatterns = "/appdepozit",
18   initParams = {
19     @WebInitParam(name = "jmxServerHost", value = "localhost")
20   }
21 )
22 public class DepozitServlet extends HttpServlet {
23   MBeanServerConnection cs=null;
24   ObjectName mbeanObjectName=null;
25   String host;
26   String port="1099";
27   String server="server";

29   public void init(ServletConfig config) {
30     try {
31       super.init(config);
32       host=config.getInitParameter("jmxServerHost");
33       String surl="service:jmx:rmi:///jndi/rmi://"+host+":"+port+"/"+server;
34       //String surl="service:jmx:iiop:///jndi/iiop://"+host+":"+port+"/"+server;
35       JMXServiceURL url = new JMXServiceURL(surl);
36       JMXConnector jmxc = JMXConnectorFactory.connect(url,null);
37       cs = jmxc.getMBeanServerConnection();
38       String domain = cs.getDefaultDomain();
39       String className="Cont";
40       String sObjectName=domain+":type="+className;
41       mbeanObjectName = new ObjectName(sObjectName);
42       cs.createMBean(className,mbeanObjectName, null, null);
43     }
44     catch(Exception e){
45       System.out.println(e.getMessage());
46       System.exit(0);
47     }
48   }
```

```
50  public void doGet(HttpServletRequest req,HttpServletResponse res)
51      throws ServletException,IOException{
52    ServletOutputStream out=res.getOutputStream();
53    String oper=req.getParameter("oper");
54    String message="";
55    double suma=0;
56    if(!oper.equals("con")){
57      String s=req.getParameter("suma");
58      suma=Double.parseDouble(s);
59    }
60    Double objValue=null;
61    try{
62      objValue=(Double)cs.getAttribute(mbeanObjectName,"Cont");
63    }
64    catch(Exception e){
65      message="JMX-Error : "+e.getMessage();
66    }
67    double value=objValue.doubleValue();
68    double x;
69    Attribute curs=null;
70    switch(oper){
71      case "dep":
72        x=value+suma;
73        curs=new Attribute("Cont", x);
74        try{
75          cs.setAttribute(mbeanObjectName,curs);
76          message="S-a depus suma";
77        }
78        catch(Exception e){
79          message="JMX-Error : "+e.getMessage();
80        }
81        break;
82      case "ext":
83        if(value>=suma){
84          x=value-suma;
85          curs=new Attribute("Cont", x);
86          try{
87            cs.setAttribute(mbeanObjectName,curs);
88            message="S-a extras suma";
89          }
90          catch(Exception e){
91            message="JMX-Error : "+e.getMessage();
92          }
93        }
94        else{
95          message="Cererea nu poate fi indeplinita";
96        }
97        break;
98      case "con":
99        message="Suma din cont este "+value+" unit.";
100       break;
101   }
102   res.setContentType("text/html");
103   out.println("<html>");
104   out.println("<head><title>Depozit</title></head>");
105   out.println("<body>");
106   out.println("<h1>Operatiuni Cont</h1>");
107   out.println("<p>");
```

```
108      out.println( message);
109      out.println("</p>");
110      out.println("</body></html>");
111      out.close();
112    }

114    public void doPost(HttpServletRequest req,HttpServletResponse res)
115       throws ServletException,IOException{
116      doGet(req,res);
117    }
118  }
```

Pagina de apelare a servlet-ului fiind (*index.html*)

```
1  <html>
2    <head>
3      <title> Servlet-ul Hello </title>
4    </head>
5    <body bgcolor="#aaeeaa">
6      <center>
7      <h1> Pagina de &#238;ntre&#355;inere a depozitului </h1>
8      <form method="post"
9            action="http://localhost:8080/appcont/appdepozit">
10       <p>Introduce&#355;i :
11       <p>Opera&#355;ia: <select name="oper" >
12             <option value="dep">Depunere
13             <option value="ext">Extragere
14             <option value="con">Consultare
15       </select>
16       <p>
17       <input type="text" name="suma" value="0">
18       <p>
19       <input type="submit" value="Executa">
20     </form>
21     </center>
22   </body>
23 </html>
```

Clientlul care urmăreşte de la distanţa contul are codul

```
1  import javax.management.ObjectName;
2  import javax.management.MBeanInfo;
3  import javax.management.MBeanAttributeInfo;
4  import javax.management.MBeanConstructorInfo;
5  import javax.management.MBeanOperationInfo;
6  import javax.management.MBeanNotificationInfo;
7  import javax.management.MBeanServerConnection;
8  import javax.management.Notification;
9  import javax.management.NotificationListener;
10 import javax.management.AttributeChangeNotification;
11 import javax.management.remote.JMXServiceURL;
12 import javax.management.remote.JMXConnector;
13 import javax.management.remote.JMXConnectorFactory;

15 public class ClientNotif {
16   public static void main(String[] args) {
17     String host="localhost";
18     String port="1099";
19     if(args.length==0){
```

```
20        System.out.println("The Server name is required");
21        System.exit(0);
22      }
23      if(args.length>=2)   host=args[1];
24      if(args.length>=3)   port=args[2];
25      ClientNotif obj=new ClientNotif();
26      try {
27        // Crearea unui conector si a obiectului de
28        // tip MBeanServerConnection
29        String surl="service:jmx:rmi:///jndi/rmi://" +
30          host+":"+port+"/"+args[0];
31        JMXServiceURL url = new JMXServiceURL(surl);
32        JMXConnector jmxc = JMXConnectorFactory.connect(url,null);
33        MBeanServerConnection cs = jmxc.getMBeanServerConnection();

35        String domain = cs.getDefaultDomain();
36        System.out.println("DefaultDomain : " +domain);
37        // Crearea obiectului OBjectName atasat MBean-ului Cont
38        String className="Cont";
39        String sObjectName=domain+":type="+className;
40        ObjectName mbeanObjectName = new ObjectName(sObjectName);

42        MBeanInfo info=cs.getMBeanInfo(mbeanObjectName);
43        getMBeanResources(info);

45        // Utilizarea notificarii
46        // Crearea unui ascultator
47        ClientListener listener = new ClientListener();
48        // Activarea notificatorului
49        cs.addNotificationListener(mbeanObjectName,listener,null,obj);

51        Thread.sleep(500);
52        // Disponibilizarea ascultatorului de notificari
53        System.out.println("Press Enter to finish !");
54        try{
55          System.in.read();
56        }
57        catch(java.io.IOException e){}
58        cs.removeNotificationListener(mbeanObjectName, listener);
59        // Disponibilizarea obiectului MBeanObjectName
60        cs.unregisterMBean(mbeanObjectName);
61      }
62      catch(Exception e) {
63        System.out.println(e.getMessage());
64        e.printStackTrace();
65      }
66    }

68    private static void getMBeanResources(MBeanInfo info){. . .}
69  }

71  class ClientListener implements NotificationListener {
72    public void handleNotification(Notification notification,
73        Object handback){
74      System.out.println("\nReceived notification: " + notification);
75      AttributeChangeNotification myNotif=
76        (AttributeChangeNotification)notification;
77      System.out.println("Sold initial : " +
78        myNotif.getOldValue().toString());
```

```
79      System.out.println("Sold curent   : " +
80        myNotif.getNewValue().toString());
81    }
82  }
```

Serverul MBean este cel prezentat în Exemplul 7.1.5.

Serverul MBean este plasat în servlet (în catalogul `WEB-INF/classes` al aplicaţiei) iar clientul care urmăreşte de la distanţa se poate afla oriunde.

După instalarea servlet-ului se lansează pe maşina acestuia server-ul MBean. Deoarece notificarea presupune existenţa MBean-ului, iar acesta se instanţează prin metoda `init` a servlet-ului este nevoie de apelarea servlet-ului, depunând 0 unităţi. După această operaţie se lansează în execuţie programul *ClientNotif.*

Întrebări recapitulative

1. Ce posibilitate oferă Java Management Extensions (JMX) ?
2. Care este structute unui MBean standard ?
3. Precizaţi conţinutul unui server MBean cu agenţi la distanţă.
4. Precizaţi termenul de notificare în legătură cu MBean.

Partea V

ANEXE

Appendix A

JavaScript Object Notation - JSON

JSON oferă o modalitate simplă (mai simplă chiar decât XML) pentru schimbul de date dintre un server şi un client.

Pentru reprezentarea datelor în JSON se utilizează structurile de date:

- colecţie de atribute, adică perechi (nume, valoare). Într-o colecţie numele atributelor trebuie să fie distincte. O colecţie de atribute este denumit obiect JSON.
- şir de valori.

Aceste structuri de date sunt prezente în toate limbajele de programare de uz general.

O colecţie de atribute se reprezintă prin

```
{numeAtribut:valAtribut,numeAtribut:valAtribut,...}
```

Un şir de valori se reprezintă prin

```
[valoare,valoare,...]
```

valAtribut, valoare poate fi un string, număr, true, false, null, o colecţie sau un şir.

JSON în JavaScript

Utilizarea entităţilor JSON în javascript este exemplificat în aplicaţia următoare.

Exemplul A.0.9

```
<HTML>
  <HEAD>
    <TITLE>Primul exemplu JavaScript</TITLE>
    <SCRIPT LANGUAGE="JavaScript">
    <!--
      var myJSONObj=[{"disciplina":"Analiza Numerica"},
      {"disciplina":"Programare distribuita"},
      {"disciplina":"Soft matematic"}];
      for(var i=0;i<myJSONObj.length;i++){
        document.writeln("<br>");
        document.writeln(myJSONObj[i].disciplina);
      }
      document.writeln("<br>");
      var myObj=eval(myJSONObj);
      document.writeln(myObj.toString());
      for(var i=0;i<myObj.length;i++){
        document.writeln("<br>");
        document.writeln(myObj[i].disciplina);
      }
    //-->
    </SCRIPT>
  </HEAD>
  <BODY>
  </BODY>
</HTML>
```

JSON în Java

google-gson

Analogul aplicaţiei javascript de mai sus, poate fi

Exemplul A.0.10

```
import com.google.gson.Gson;
import com.google.gson.reflect.TypeToken;
import java.lang.reflect.Type;
import java.util.Collection;
import java.util.Iterator;

class Disciplina{
  private String nume;
  Disciplina(){}

  Disciplina(String nume){
    this.nume=nume;
  }
  public String getNume(){
    return nume;
  }
}
```

```
19 public class TestGSON{
20   public static void main(String[] args){
21     Gson gson=new Gson();
22     Disciplina an=new Disciplina("Analiza numerica");
23     Disciplina pd=new Disciplina("Programare distribuita");
24     Disciplina sm=new Disciplina("Soft matematic");
25     Disciplina[] discipline={an,pd,sm};
26     String json=gson.toJson(discipline);
27     System.out.println(json);
28     Type collectionType = new TypeToken<Collection<Disciplina>>(){}.getType();
29     Collection<Disciplina> d = gson.fromJson(json,collectionType);
30     Iterator<Disciplina> iter=d.iterator();
31     while(iter.hasNext()){
32       Disciplina dis=iter.next();
33       System.out.println(dis.getNume());
34     }
35   }
36 }
```

javax.json

Interfaţa `javax.json.JsonValue` introduce obiectele nemodificabile (*immutable*) `JsonArray, JsonObject, JsonString, JsonNumber, JsonValue.TRUE, JsonValue.FALSE, JsonValue.NULL.`

Obiectele se instanţiază prin intermediul unor metode statice ale clasei `javax.json.Json.`

Există structura de interfeţe

`JsonValue`	`JsonStructure`	`JsonArray`
		`JsonObject`
	`JsonString`	
	`JsonNumber`	acoperă tipurile de date Java numerice `BigDecimal, BigInteger, int, long, double`

Clasa `javax.json.Json`

Metode

- `static JsonObjectBuilder createObjectBuilder()`
- `static JsonArrayBuilder createArrayBuilder()`
- `static JsonWriter createWriter(java.io.Writer` *writer*`)`
- `static JsonReader createReader(java.io.Reader` *reader*`)`

Interfaţa `javax.json.JsonObjectBuilder`

Metode

- `JsonObjectBuilder add(String` *name, TipJson value)*

  TipJson∈{`BigDecimal, BigInteger, int, long, double, boolean, JsonObjectBuilder, JsonArrayBuilder, JsonValue, String`}.

- `JsonObjectBuilder addNull()`
- `JsonObject builder()`

Şablon de utilizare

```
JsonObject jsonObject=Json.createObjectBuilder()
    .add("name", value)
    . . .
    .build();
```

Interfaţa `javax.json.JsonArrayBuilder`

Metode

- `JsonArrayBuilder add(`*TipJson value)*

  TipJson∈{`BigDecimal, BigInteger, int, long, double, boolean, JsonObjectBuilder, JsonArrayBuilder, JsonValue, String`}.

- `JsonArrayBuilder addNull()`
- `JsonArray builder()`

Şablon de utilizare

```
JsonArray jsonArray=Json.createArrayBuilder()
    .add(value)
    . . .
    .build();
```

Interfaţa `javax.json.WriterBuilder`

Metode

- `void writeArray(JsonArray` *array)*
- `void writeObject(JsonObject` *object)*
- `void close()`

Şablon de utilizare

```
PrintWriter printWriter=new PrintWriter(System.out)
JsonWriter jsonWriter=Json.createWriter(printWriter);
jsonWriter.writeArray(jsonArray);
jsonWriter.close();
```

Interfaţa `javax.json.ReaderBuilder`

Metode

- `void readArray()`
- `void readObject()`
- `void close()`

Şablon de utilizare

```
String string=. . .
JsonReader jsonReader = Json.createReader(new StringReader(string));
JsonArray array = jsonReader.readArray();
jsonReader.close();
```

Exemplul A.0.11 *Crearea unui fişier json.*

```
1  import javax.json.JsonArray;
2  import javax.json.JsonArrayBuilder;
3  import javax.json.JsonObject;
4  import javax.json.JsonObjectBuilder;
5  import javax.json.JsonWriter;
6  import javax.json.Json;
7  import java.io.PrintWriter;
8  import java.io.IOException;

10 public class GenerateJSON{
11   public static void main(String[] args){
12     JsonArray jsonArray=Json.createArrayBuilder()
13         .add(Json.createObjectBuilder()
14             .add("nume","Analiza numerica"))
15         .add(Json.createObjectBuilder()
16           .add("nume","Programare distribuita"))
17         .add(Json.createObjectBuilder()
18           .add("nume","Soft matematic"))
19         .add(100)
20         .add("javax.json")
21         .add(Json.createArrayBuilder()
22           .add(1)
23           .add(2)
24           .add(3))
25         .add(Json.createArrayBuilder()
26           .add(4)
27           .add(5)
28           .add(6))
29       .build();
30     System.out.println("System.out : "+jsonArray);
31     String fileName="exemplu.json";

33     try{
34       JsonWriter jsonWriter=Json.createWriter(new PrintWriter(fileName));
35       jsonWriter.writeArray(jsonArray);
36       jsonWriter.close();
37     }
```

```
38      catch(Exception e){
39        System.out.println(e.getMessage());
40      }
41      JsonWriter jsonWriter=Json.createWriter(new PrintWriter(System.out));
42      jsonWriter.writeArray(jsonArray);
43      jsonWriter.close();
44    }
45  }
```

Exemplul A.0.12 *Consultarea fişierului json creat în exemplul anterior.*

```
1  import javax.json.JsonArray;
2  import javax.json.JsonObject;
3  import javax.json.JsonReader;
4  import javax.json.Json;
5  import javax.json.JsonValue;
6  import javax.json.JsonString;
7  import javax.json.JsonNumber;
8  import java.io.FileReader;
9  import java.io.IOException;
10 import java.util.Iterator;
11 import java.util.Map;
12 import java.util.Set;

14 public class ReadJSON{
15   public static void main(String[] args){
16     String fileName="exemplu.json";
17     String path="d:\\mk\\DISTR2\\GSON\\JEE\\ex1\\";
18     JsonArray array=null;
19     try{
20       JsonReader jsonReader =
21         Json.createReader(new FileReader(path+fileName));
22       array = jsonReader.readArray();
23       jsonReader.close();
24     }
25     catch(IOException e){
26       System.out.println("Ex : "+e.getMessage());
27     }
28     analyse(array);
29   }

31   private static void analyse(JsonArray v){
32     Iterator<JsonValue> iterator=v.iterator();
33     while(iterator.hasNext()){
34       JsonValue value=(JsonValue)iterator.next();
35       if(value instanceof JsonArray){
36         JsonArray array=(JsonArray)value;
37         analyse(array);
38       }
39       if(value instanceof JsonObject){
40         JsonObject obj=(JsonObject)value;
41         analyseJsonObject(obj);
42       }
43       if(value instanceof JsonString){
44         JsonString string=(JsonString)value;
45         String s=string.getString();
46         System.out.println(s);
```

```
47          }
48          if(value instanceof JsonNumber){
49            JsonNumber number=(JsonNumber)value;
50            double d=number.doubleValue();
51            System.out.println(d);
52          }
53        }
54      }

56      private static void analyseJsonObject(JsonObject obj){
57        Map<String,JsonValue> object=(Map)obj;
58        Set<String> keys=object.keySet();
59        Iterator<String> iter=keys.iterator();
60        while(iter.hasNext()){
61          String name=iter.next();
62          System.out.println();
63          System.out.println("JsonObject name : "+name);
64          JsonValue vv=(JsonValue)object.get(name);
65          if(vv instanceof JsonArray){
66            JsonArray array=(JsonArray)vv;
67            analyse(array);
68          }
69          if(vv instanceof JsonObject){
70            JsonObject o=(JsonObject)vv;
71            analyseJsonObject(o);
72          }
73          if(vv instanceof JsonString){
74            JsonString string=(JsonString)vv;
75            String s=string.getString();
76            System.out.println(s);
77          }
78          if(vv instanceof JsonNumber){
79            JsonNumber number=(JsonNumber)vv;
80            double d=number.doubleValue();
81            System.out.println(d);
82          }
83        }
84      }
85  }
```

Appendix B

Simple Object Access Protocol - SOAP

SOAP - Simple Object Access Protocol - este un protocol de comunicaţii între aplicaţii. În prezent SOAP este protocolul standard pentru servicii Web prin Internet. SOAP este independent de platforma de calcul şi de limbajul de programare.

SOAP se bazează pe XML (eXtensible Markup Language) şi este un standard W3C (World Wide Web Consortium).

B.1 Mesaje SOAP

Un mesaj SOAP este un document XML constând din

- o învelitoare (*envelope*) care poate conţine
- un număr arbitrar de antete (*header*);
- un corp (*body*);
- un număr variabil de obiecte ataşate (*attachments*) MIME (Multipurpose Internet Mail Exchange).

Astfel un mesaj SOAP apare sub forma documentului XML

```
<?xml version="1.0" encoding="UTF-8"?>
<Envelope xmlns="http://schemas.xmlsoap.org/soap/envelope/">
   <Header/>
   <Body>
```

```
      <Fault/>
   </Body>
</Envelope>
```

Facilităţile Java de manipulare a mesajelor SOAP sunt conţinute în pachetul `javax.xml.soap`, din distribuţia jdk.

Crearea unui mesaj SOAP

```
MessageFactory mf=MessageFactory.newInstance();
SOAPMessage soapMsg=mf.createMessage();
```

Mesajul creat are definită structura de bază a mesajului SOAP: invelitoarea, un antet şi corp. Aceste elemente pot fi accesate prin

```
SOAPPart part=soapMsg.getSOAPPart();
SOAPEnvelope envelope=part.getEnvelope();
SOAPHeader header=envelope.getHeader();
SOAPBody body=envelope.getBody();
```

Completarea corpului unui mesaj SOAP

Oricărui element în este asociat un obiect `javax.xml.soap.Name`. Din acest obiect se pot afla

- `String getLocalName()`
- `String getPrefix()`
- `String getURI()`
- `String getQualifiedName()`

Numele calificat are structura

```
<prefix:NumeleLocal xmlns:prefix="uri">
```

Elementul cu numele local "e1" se generează prin

```
Name n1=envelope.createName("e1");
```

În *body* un element se poate include, pe baza numelui creat prin

```
SOAPElement e1=body.addBodyElement(n1);
```

În general, un element se include în elementul părinte, care poate fi chiar şi *body,* prin metoda clasei `SOAPElement`

`SOAPElement addChildElement(Name` *name*`)`

Completăm elementul *e1* cu un *text: "primul"*, prin

```
e1.addTextNode("primul");
```

Exemplul B.1.1 *Mesajul SOAP*

```
<SOAP-ENV:Envelope
   xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/">
<SOAP-ENV:Header/>
<SOAP-ENV:Body>
   <e1>
      primul
      <e11>
          al treilea
      </e11>
   </e1>
   <e2>
      al doilea
   </e2>
</SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

se obţine cu programul

```
import javax.xml.soap.MessageFactory;
import javax.xml.soap.SOAPMessage;
import javax.xml.soap.SOAPPart;
import javax.xml.soap.SOAPEnvelope;
import javax.xml.soap.SOAPBody;
import javax.xml.soap.SOAPElement;
import javax.xml.soap.Name;
import java.io.FileOutputStream;

public class MsgSOAP{
  public static void main(String[] args){
    Name name=null;
    try{
      MessageFactory mf=MessageFactory.newInstance();
      SOAPMessage soapMsg=mf.createMessage();
      SOAPPart part=soapMsg.getSOAPPart();
      SOAPEnvelope envelope=part.getEnvelope();
      SOAPBody body=envelope.getBody();
      Name n1=envelope.createName("e1");
      SOAPElement e1=body.addBodyElement(n1);
      e1.addTextNode("primul");
      Name n2=envelope.createName("e2");
      SOAPElement e2=body.addBodyElement(n2);
      e2.addTextNode("al doilea");
      Name n11 =envelope.createName("e11");
      SOAPElement e11=e1.addChildElement(n11);
      e11.addTextNode("al treilea");
```

```
28        FileOutputStream f=new FileOutputStream("MySOAPMessage.xml");
29        soapMsg.writeTo(f);
30      }
31      catch(Exception e){
32        System.out.println("Exception : "+e.getMessage());
33      }
34    }
35  }
```

Un mesaj SOAP se poate salva într-un fişier text cu

```
FileOutputStream f=new FileOutputStream(. . .);
soapMsg.writeTo(f);
```

Conţinutul fişierului este

```
SOAP-ENV:Envelope xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/">
<SOAP-ENV:Header/><SOAP-ENV:Body><e1>primul<e11>al treilea</e11></e1>
<e2>al doilea</e2></SOAP-ENV:Body></SOAP-ENV:Envelope>
```

Preluarea elementelor din corpul unui mesaj SOAP

```
SOAPBody body=. . .
SOAPBodyElement element=null;
Iterator iterator=body.getChildElements();
   while(iterator.hasNext()){
      element=(SOAPBodyElement)iterator.next();
      String name=element.getElementName();
      if(name.getLocalName().equals("numeCamp")){
         String s=element.getValue();
            . . .
      }
   }
```

```
1  import javax.xml.soap.MessageFactory;
2  import javax.xml.soap.SOAPBody;
3  import javax.xml.soap.SOAPBodyElement;
4  import javax.xml.soap.SOAPMessage;
5  import javax.xml.soap.SOAPEnvelope;
6  import javax.xml.soap.SOAPPart;
7  import javax.xml.soap.SOAPElement;
8  import javax.xml.soap.Name;
9  import java.util.Iterator;
10 import java.io.FileInputStream;
11 import org.w3c.dom.Node;

13 public class MsgSOAPReceiver{

15   public static void analyze(SOAPElement rootElement){
```

```
16      Iterator iterator=rootElement.getChildElements();
17      while(iterator.hasNext()){
18        SOAPElement element = (SOAPElement) iterator.next();
19        short nodeType=element.getNodeType();
20        System.out.println(nodeType);
21        Name name=element.getElementName();
22        System.out.println("name : "+name.getLocalName());
23        System.out.println("value : " + element.getValue());
24        if(nodeType==Node.ENTITY_NODE)analyze(element);
25      }
26    }

28    public static void main(String[] args) {
29      try {
30        FileInputStream fis=new FileInputStream("MySOAPMessage.xml");
31        MessageFactory mf = MessageFactory.newInstance();
32        SOAPMessage soapMsg = mf.createMessage(null,fis);
33        SOAPPart part=soapMsg.getSOAPPart();
34        SOAPEnvelope envelope=part.getEnvelope();
35        SOAPBody body=envelope.getBody();
36        analyze(body);
37      }
38      catch (Exception ex) {
39        ex.printStackTrace();
40      }
41    }
42  }
```

Observaţia B.1.1

Utilizarea mesajelor SOAP în serviciile Web bazate pe JAX-WS este complet transparentă programatorului.

Observaţia B.1.2

Produsul *Oracle-Open Message Queue* oferă posibilitatea transformării unui mesaj SOAP în mesaj JMS şi invers.

Transformarea unui mesaj SOAP în mesaj JMS

```
SOAPMessage soapMsg=. . .
Message msg=
   MessageTransformer.SOAPMessageIntoJMSMessage(soapMsg,session);
```

Transformarea unui mesaj JMS în mesaj SOAP

```
MessageFactory mf=. . .
Message msg=. . .
SOAPMessage soapMsg =
    MessageTransformer.SOAPMessageFromJMSMessage(msg,mf);
```

Bibliografie

[1] ATHANASIU I., COSTINESCU B., DRĂGOI O.A., POPOVICI F.I., 1998, *Limbajul Java. O perspectivă pragmatică.* Ed. Computer Libris Agora, Cluj-Napoca.

[2] BOIAN F.M., BOIAN R. F., 2004, *Tehnologii fundamentale Java pentru aplicaţii Web.* Ed. Albastră, Cluj-Napoca.

[3] BOIAN F.M., 2011, *Servicii Web; Modele, Platforme, Aplicaţii.* Ed. Albastră, Cluj-Napoca.

[4] BURAGA S.C., 2001, *Tehnologii Web.* Ed. Matrix Rom,Bucureşti.

[5] BURAGA S. (ed), 2007, *Programarea în Web 2.0.*, Ed. Polirom, Iaşi.

[6] JURCĂ I., 2000, *Programarea reţelelor de calculatoare.* Ed. de Vest, Timişoara.

[7] HUNTER J., CRAWFORD W., 1998, *Java Servlet Programming.* O'Reilly.

[8] ALBOAIE L., BURAGA S., 2006, *Servicii Web.* Ed. Polirom, Iaşi.

[9] SCHEIBER E., 2007, *Programare concurentă şi paralel distribuită în Java.* Ed. Albastră, Cluj-Napoca.

[10] TANASĂ Ş., ANDREI Ş., OLARU C., 2011, *Java de la 0 la extert.* Ed. Polirom, Iaşi.

[11] TANASĂ Ş., OLARU C., 2005, *Dezvoltarea aplicaţiilor Web folosind Java.* Ed. Polirom, Bucureşti.

[12] * * * , Java 2 SDK 1.*.*/docs/, Sun Microsystems.

[13] * * * , JavaWS Tutorial 1.*, Sun Microsystems.

[14] * * * , J2EE Tutorial 1.5, Sun Microsystems.

[15] * * * , Java 2 Tutorial, Sun Microsystems.